COBOL report program generation by macro processor

Thesis

How to cite:

For guidance on citations see FAQs.

© 1981 The Author

https://creativecommons.org/licenses/by-nc-nd/4.0/

Version: Version of Record

Link(s) to article on publisher’s website:
http://dx.doi.org/doi:10.21954/ou.ro.00010108

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online’s data policy on reuse of materials please consult the policies page.
SHEILA STEVENS B.Sc.

COBOL REPORT PROGRAM GENERATION BY MACRO PROCESSOR

Ph.D. THESIS

COMPUTER SCIENCE

FACULTY OF MATHEMATICS

FEBRUARY 1981

Date of submission: 19-2-81
Date of award: 15-7-81
ABSTRACT

The research described is directed towards the development of an interactive COBOL program generator. This aims to simplify computer usage and contribute to the more effective use of computer resources, by alleviating the burdensome, time-consuming and error-prone activity of programming.

The work is based on PG/1, an existing macro processor, and in particular considers the generation of report programs.

The problems of program generation by macro processor are first explored using Filetab, an existing report specification language, and avoiding the interactive aspect. This establishes the feasibility of generating COBOL programs but identifies shortcomings in the macro processor facilities. Enhancements to the macro processor are defined and the PG/2 version is created for use.

The development of an interactive self-instructional COBOL report program generator for the casual user is chosen as the area for deeper investigation.

From a consideration of data bases and query languages it is concluded that a relational view of the data forms the most natural basis for a computer-dominated, non-procedural report-specifying dialogue with a casual user.

A design study for an interactive self-teaching COBOL report program generator, based on the PG/2 macro processor and offering some of the benefits of a Relational Data base Management System, is described. The validation of the user's responses is a major undertaking and leads to a large system. Details of the macros implementing selected features are presented.
It is concluded that program generation appears efficient but may be of even greater use to a professional programmer than to the non-specialist user. The PG/2 macro processor, although not an ideal tool for COBOL program generation, serves to identify the characteristics of an appropriate tool. There is potential for extending and adapting the techniques used to other applications.
I wish to record my gratitude to the Computer Centre of
Queen Mary College, London University, the University of
Southampton Computing Service and the Open University Student
Computing Service for the use of computer facilities.

My thanks are due also to the management of F International
Limited for permission to make use of their confidential estimating
procedures.

I am also indebted to Mr. Adam J. Gawronski of the Open
University Student Computing Service for maintaining the PG/1
Macro Processor and for implementing the enhancements for the
PG/2 version.

Above all I must thank Professor R. M. Pengelly for his
guidance and supervision throughout this project.

Winchester

February, 1981.
## CONTENTS

1. **INTRODUCTION**
   - 1.1 Background to the Project  
   - 1.2 Aims  
   - 1.3 Approach  
   - 1.4 Results and conclusions  
   - 1.5 Outline of Thesis  

2. **BACKGROUND TO THE PRELIMINARY STUDY**
   - 2.1 Introduction  
   - 2.2 PG/1 Macro Processor  
   - 2.3 Filetab  

3. **DEVELOPMENT OF A FILETAB TO COBOL PROGRAM GENERATOR**
   - 3.1 Introduction  
   - 3.2 Initial approach  
   - 3.3 Storing and sequencing generated COBOL statements  
   - 3.4 The design of the COBOL program generator  
   - 3.5 Example of the break-down of a Filetab directive into COBOL statements  
   - 3.6 The macro processing of Filetab directives  
   - 3.7 Assessment of the Filetab to COBOL program generator  
   - 3.8 Summary, appraisal and conclusions  

4. **ENHANCEMENTS FOR THE PG/1 MACRO PROCESSOR**
   - 4.1 Introduction  
   - 4.2 Reduction of core storage demands  
   - 4.3 Reduction in the use of labels  
   - 4.4 Access to subfiles  
   - 4.5 Emptying the output stack  
   - 4.6 String handling facilities  
   - 4.7 Other facilities  
   - 4.8 Assessment of the PG/2 Macro Processor
5. THE USER INTERFACE

5.1 Introduction
5.2 User's view of the data
5.3 Relational approach
5.4 Dialogue design
5.5 Specification of conditions for data retrieval
5.6 Report layout design
5.7 Own code processing
5.8 User feedback
5.9 Definition of a model problem
5.10 Summary and appraisal

6. SYSTEM SPECIFICATION

6.1 Introduction
6.2 System overview
6.3 The catalogue system
6.4 Input processing
6.5 Report output facilities
6.6 Data manipulation facilities
6.7 Summary and appraisal

7. IMPLEMENTATION OF SELECTED FEATURES

7.1 Introduction
7.2 Outline structure
7.3 Subfile usage
7.4 Create an empty Catalogue
7.5 Change the Catalogue password
7.6 Inserting and/or deleting Catalogue file descriptions
7.7 Use of routines
7.8 Stage 1 - Introduction
7.9 Stage 2 - Password dialogue
7.10 Stage 3 - User's data base submodel
7.11 Stage 4 - Selection of problem domains
7.12 Stage 5 - Temporary extensions of the data base
7.13 Stage 6 - Data base inconsistencies
7.14 Stage 7 - Selection of data for retrieval
7.15 Stage 8 - Report layout introduction
7.16 Stage 9 - Editing
7.17 General strategy for processing report output format specifications
7.18 Stage 10 - Detail line(s) specification
7.19 Stage 11 - Extra data items - date, time, page number
7.20 Stage 12 - Report title specification
7.21 Stage 13 - Page heading specification
7.22 Stage 14 - Sequence break heading specification
7.23 Stage 15 - Subtotal line(s) specification
7.24 Stage 16 - Total line(s) specification
7.25 Stage 17 - Sample page dialogue
7.26 Stage 18 - Own code processing
7.27 HELP macro
7.28 Generation of a complete COBOL program

8. APPRAISAL AND CONCLUSIONS
8.1 Introduction
8.2 Relational view of the data
8.3 Security procedures
8.4 User's approach to the system
8.5 Instructional text
8.6 Magnitude of the COBOL generating system
8.7 The generated COBOL program
8.8 Towards more effective computer usage
8.9 Aspects for further study
8.10 Concluding summary
<table>
<thead>
<tr>
<th>Figure</th>
<th>Illustration Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Report layout for Filetab sample</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>Contributions by Filetab directives to COBOL program divisions</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>The three phase system using data subfiles</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>The three phase system using 'grown' macros</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Filetab directive contributions to 'grown' macros and the generated COBOL program</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>Outline of processing in the generated COBOL program</td>
<td>43</td>
</tr>
<tr>
<td>3.6</td>
<td>Outline processing for Phase 1 of the Filetab to COBOL program generator</td>
<td>48</td>
</tr>
<tr>
<td>3.7</td>
<td>Outline processing of Filetab &quot;INL parameters&quot;</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>STAFF Relation</td>
<td>91</td>
</tr>
<tr>
<td>5.2</td>
<td>PERSONNEL Relation</td>
<td>91</td>
</tr>
<tr>
<td>5.3</td>
<td>Join of STAFF and PERSONNEL Relations</td>
<td>91</td>
</tr>
<tr>
<td>5.4</td>
<td>Limited entry decision table format</td>
<td>98</td>
</tr>
<tr>
<td>5.5</td>
<td>Sketch of the required report layout</td>
<td>110</td>
</tr>
<tr>
<td>6.1</td>
<td>Report program generating system</td>
<td>118</td>
</tr>
<tr>
<td>6.2</td>
<td>Catalogue structure</td>
<td>124</td>
</tr>
<tr>
<td>6.3</td>
<td>Librarian macros for catalogue maintenance</td>
<td>131</td>
</tr>
<tr>
<td>6.4</td>
<td>COBOL report generating system input summary</td>
<td>133</td>
</tr>
<tr>
<td>6.5</td>
<td>Record category and field type summary</td>
<td>138</td>
</tr>
<tr>
<td>6.6</td>
<td>Editing examples</td>
<td>150</td>
</tr>
<tr>
<td>7.1</td>
<td>Outline structure of the COBOL report program generator</td>
<td>166</td>
</tr>
<tr>
<td>7.2</td>
<td>Macro linkages for the stages of the problem specifying dialogue</td>
<td>169</td>
</tr>
<tr>
<td>7.3</td>
<td>Outline processing for the CREATE macro</td>
<td>176</td>
</tr>
<tr>
<td>7.4</td>
<td>Outline processing for the PASSCHANGE macro</td>
<td>179</td>
</tr>
<tr>
<td>7.5</td>
<td>Outline processing for the LIBRARIAN macro</td>
<td>182</td>
</tr>
<tr>
<td>7.6</td>
<td>Outline processing for the LIBPRELIM macro</td>
<td>184</td>
</tr>
</tbody>
</table>
APPENDICES

I  PG/1 MACRO PROCESSOR
II  FILETAB LANGUAGE FEATURES
III  MACRO PROCESSING OF FILETAB DIRECTIVES
IV  PG/2 MACRO PROCESSOR
V  FORMATS OF COMMON DATA SUBFILES
VI  LANGUAGE FACILITIES FOR COBOL GENERATION
VII  COBOL REPORT GENERATING SYSTEM MACRO PROCESSING DETAILS
VIII  COBOL REPORT PROGRAM GENERATION DETAILS
IX  DIALOGUE SAMPLES
X  LISTING OF GENERATED COBOL PROGRAM
REFERENCES


[18] COBRA, COBOL Generation System, ICL Dataskil Ltd.


[22] F INTERNATIONAL Ltd., The Bury, Church Street, Chesham, Bucks.


SEPARATE FOLDER OF LISTINGS

CONTENTS

1. OUTLINE PROCESSING FOR THE COBOLGEN MACRO
2. INITIALISATION SECTION OF COBOLGEN MACRO
3. SKELETON COBOL PROGRAM - GOPART2 SUBFILE
4. * COMMENT PROCESSING
5. *FILE PROCESSING
6. *INL PROCESSING
7. *TITLE PROCESSING
8. *HEAD PROCESSING
9. *OUT PROCESSING
10. *GO PROCESSING
11. MESSAGES AND CREATE SUBFILES
12. PASSCHANGE SUBFILE
13. LIBRARIAN SUBFILE
14. LIBPRELIM SUBFILE
15. LIBDELETE SUBFILE
16. LIBADD SUBFILE
17. DIAL1 AND STAGE1 SUBFILES
18. DIAL2 AND STAGE2 SUBFILES
19. DIAL3 AND STAGE3 SUBFILES
20. STAGE3-1 SUBFILE
21. DIAL4 AND STAGE4 SUBFILES
22. DIAL5 AND STAGE5 SUBFILES
23. STAGE5-1 SUBFILE
24. STAGE5-2 SUBFILE
25. DIAL6 AND STAGE6 SUBFILES
26. DIAL7 AND STAGE7 SUBFILES
27. DIAL8 AND STAGE8 SUBFILES
28. DIALHELP AND HELP SUBFILES
CHAPTER 1

INTRODUCTION

1.1 BACKGROUND TO THE PROJECT

With the increasing number of computers in use, especially smaller ones, there is a growing demand for computer programs. This demand will soon greatly exceed the number of programmers available to undertake program development. In general computers are difficult to use especially by the non-programmer, but increasingly such people will be required to make use of a computer. There are three approaches by which a non-programmer can make use of a computer:

1. He can learn to program.
2. He can brief a programmer to develop a program to meet his requirements.
3. The computer scientist can create a system which enables the non-programmer to communicate his requirements directly to the computer.

The first approach is time consuming. The second approach may be expensive (even if a programmer is available) and there is potentially a poor interface. Between the realisation of the problem and obtaining results, both these approaches suffer delays while the program is coded and developed. Coding is often a burden, especially in a verbose language like COBOL.

The third approach is also potentially expensive and carries with it the problem of defining the 'user interface'. It does, however, have the important advantage of shortening the time between the realisation of a problem and obtaining results. It is in this latter approach that a program generator and a problem orientated language play a significant part.

A program generator is a computer software package which
will produce a program to perform a specific version of some general application, when supplied with data specifying the requirements for the particular application. The purposes of a program generator are twofold:

1. To make it 'easier' to use the computer as a tool for problem solving by providing a means of creating an appropriate user interface.

2. To enable the user to use the computer 'more efficiently'.

By making the use of a computer a more natural process, the program generator serves to shorten the time between the realisation of the need for a computer program to solve a problem and the completion of a working program. This approach should use less resources when the creation of a program is followed by its regular use. Not only does the use of a program generator assist the professional programmer to develop a program more efficiently, but it is also a step towards the goal of allowing a much larger class of users to use the computer as a problem solving tool.

Non-procedurality is essential in the design of problem orientated languages for use with program generators. That is to say, the trend must be towards systems where the user specifies the problem to be solved and not 'how' the problem should be solved. For example in a statistical package, such as EASYSTAT [24], the user describes in short English phrases the computation he wishes carried out on the data, e.g. Multiple linear regression using 20 variables. He does not have to define the formulae and processing needed to obtain the answers. Similarly with a report generator system, e.g. Filetab [7], the user has only to supply directives describing the files containing the data and the form and content of the required report. He does not have to specify how the files are to be processed to extract the data, nor how it
is to be reorganised into the form required in the report.

There are two possible approaches to non-procedural languages:

1. The creation of an interpreter to translate the statements of the high level language into machine code instructions which are immediately executed, c.f. Filetab.

2. The creation of a compiler which makes use of the overall statement of the problem and produces a machine language routine to be executed later, c.f. this project.

When dealing with a very high level language (and a non-procedural one of that) there is a need to evolve the language in the light of experience. Also the non-procedural very high level language needs to be compiled into a procedural high level language such as FORTRAN, COBOL etc.

In the creation of any problem-to-problem translator the facility to manipulate strings of characters is essential: macro processors have been shown by McIlroy [1], Halpern [2], Strachey [3] and Brown [4] to meet this requirement. A macro processor provides general purpose string manipulation facilities for language translation. It has the added advantage that extensions to the language, created to meet the user's changing needs, can be achieved by macro extension.

It has already been shown by Mandil [5] that a general purpose macro processor can be used as a powerful tool in the development of a general purpose translator. As an alternative to a special purpose interpreter, this study investigated 'problem-to-FORTRAN' generation in order to create an efficient system for solving a range of numerical problems. Under this system the problems are specified at a terminal by means of a user/system interactive dialogue. The system assumes the initiative for most
of the time and interrogates the user about his problem. The user is guided at every step and default options are provided to cover common requirements. The system relies on the existence of a good library of numerical subprograms.

In the scientific field problems tend to be characterized by a low volume of input data, complex algorithms, a considerable amount of mathematical calculation and relatively simple output formats. By contrast in the commercial field emphasis changes to a high volume of input data, straightforward algorithms, a lower level of computation and the production of printed reports and/or output files, which may have quite complex formats.

This project was initially motivated by an investigation of ways to make it 'easier' for the casual user to use the computer as a tool for solving a limited range of commercial data processing problems. Essentially the interest lies in whether ways can be found to help the user use the computer more efficiently — that is to investigate ways of reducing wastage in terms of time and resources at the man/computer interface.

The problems of communication between user, systems analyst, programmer and computer are chiefly responsible for the inefficient use of resources. The aim is to remove some of these difficulties by the creation of a system where the non-programmer can communicate a range of simple problems directly to the computer, in order to produce a program to solve his problem. Thus shortening the time span between the realisation of a problem and obtaining results.

One of the commonest problems in the commercial environment is the 'request for a report', which usually means the regular repetition of relatively long runs of relatively straightforward data processing. Another consideration is the production of
report programs for small (micro) computers. A program generator running on a large machine may be a good way to tackle this problem. It is the solution of problems like these that the COBOL generation work described in this thesis represents an exploratory step. Report program generation is a good test case because it covers a variety of important techniques. If difficulties arising in the generation of COBOL report programs from information supplied by a casual user can be overcome, then it is likely that a wider range of problems can be solved by the generation of a COBOL program.

1.2 AIMS

The object of this work on COBOL program generation has been to discover and explore some of the benefits that could arise from carefully and systematically exploiting a relatively small number of basic ideas. The key idea is that of using a computer dominated dialogue as a basis for generating a syntactically and semantically correct COBOL report program. The success of generating a syntactically correct program is readily measured, but that of generating a semantically correct program is less quantifiable.

The main aims of the work are as follows:
1. To explore the problems of COBOL generation.
2. To explore the software for dialogue based program generation.
3. To develop a computer/user dialogue, in a relatively non-procedural language, by which the casual user can describe the report he requires.
4. To develop a system, which makes use of the information gathered during the dialogue, to generate a syntactically correct program.
5. To generate COBOL programs that have an efficiency
comparable with that of hand coded programs, but which require fewer man hours and computer resources for their development.

6. As a step towards ensuring the generation of semantically correct programs, to provide some features which help check that the user's requests are sensible and allow him some opportunity to verify that the program output will meet his requirements.

7. Ultimately, to consider some other aspects of programming and COBOL program generation.

1.3 APPROACH

Although the 'request for a report' was selected as the area for deeper investigation, throughout the design stages of the project the possibility of extending the range of problems, which could be solved by the generation of a COBOL program, was born in mind.

The practical work for this project was undertaken using an ICL 1900 series computer and this greatly influenced the choice of software. The PG/1 macro processor [5] was selected for the development of the COBOL program generator for two main reasons:

1. It represented the only convenient general purpose string processing system available on the ICL 1900 series computer at the start of the project.

2. The macro processor had previously been used for a FORTRAN generation project, against which it was interesting to compare the problems encountered in the generation of COBOL.

The PG/1 macro processor was a successor to the KP/1 macro processor [6].

The project began by investigating the feasibility of, and problems encountered in generating effective COBOL programs from a normal report generator language using the PG/1 macro processor.
It was decided to use an existing report generator language as the problem specifying language, because this had the advantage that comparisons in the size and efficiency of the generated COBOL program could be made. It also permitted all the effort to be directed towards the development of the macro definitions without the burden of designing a problem specifying language. The National Computing Centre's Filetab package [7], whose ICL 1900 series version is called TABN, was an appropriate report generator language for this purpose. The Filetab language features are very comprehensive, so for the purposes of this initial study a subset was selected. These covered the simpler facilities required for generating a typical report.

The initial study showed that acceptable COBOL programs could be generated by using macro definitions to translate the report specifications. This stage of the project served to identify enhancements to the PG/1 macro processor in order to give greater manipulative power and flexibility. The necessary enhancements were defined and this led to the creation of the PG/2 macro processor, which formed the basis for the remainder of the project. (The implementation of these enhancements did not form part of the candidate's work on the project.)

A design study was then undertaken to develop a non-procedural computer/user dialogue as a means of specifying the form and content of the report. As part of this study consideration was given to current work on data bases and query languages. The aim of the project was to express the dialogue in terms of the user's view of the data. In addition to requesting answers to specific questions, the computer dialogue adopts a tutorial approach by displaying instructional text illustrated by examples. From this study it became apparent that it was possible
to develop a system incorporating some of the benefits associated with Data base Management Systems (DBMS). In particular it became clear that the user need not be aware of the actual data formats and that each user could be given his own view of the data. Although any form of data description could have been adopted, a relational language was chosen because it seems to be the more natural from the casual user's point of view. Besides providing an element of data independence it emerged that there was scope for the creation of a range of 'privacy' mechanisms.

The outline design for a complete relational 'DBMS' type system for the generation of COBOL programs to print a report for the non-programmer was prepared. The system consists of three main parts.

The first part, for use by a 'Data base Administrator', consists of macro definitions to create and maintain a Catalogue of File descriptions which describe the data in the data base. Essentially the person in charge of the system has to maintain a data dictionary which includes specifying how each class of user 'views' the files available. The files described in the Catalogue may be on various media, organised in a variety of ways and need not have been designed primarily for use in a relational data base. The system provides for the creation of a privacy mechanism to protect the data when it is accessed by the generated COBOL program.

The second part of the system is self-tutorial in approach and gathers, by means of a computer/user dialogue, information about the report required by the non-programmer user. This information is stored in ways which enables the third part of the system to generate the COBOL program to produce the report the user requires. As well as being self-tutorial, the second part of
the system is designed to incorporate a 'help' facility to enable the user to change his mind about an earlier decision and to experiment with report layout designs. The system also permits the user to view a sample of computer/user dialogue which covers a situation similar to that at the point where he requested help. Throughout the dialogue there are many instances where the user may opt for the default reply to a question and thus reduce the amount of information to be keyed in.

The maintenance of the data base Catalogue and the computer/user dialogue are designed as foreground jobs to be run at a terminal. The generation of the COBOL report program, its compilation and execution and the maintenance of the data base files are designed to run in background mode under a batch operating system.

The technical effort required to implement a fully operational system would greatly exceed the time and facilities available for practical work with this sort of project. The practical work for this stage of the project was therefore directed towards the implementation of selected key features and served three main purposes:

1. To validate aspects of the candidate's ideas and system design.
2. To explore the problems with dialogue creation and program generation using the macro processor.
3. To establish the validity of the generated COBOL programs.
1.4 RESULTS AND CONCLUSIONS

Although a fully operational system is needed to validate the potential of the approach adopted, the project shows that a dialogue can be created which can be used to generate COBOL programs. The validation of the user's responses is a major undertaking leading to a large system. Program generation appears efficient but may be of even more use to the professional programmer than to the casual user. The macro processor, although not an ideal tool for the generation of COBOL programs, served to discover the characteristics of an appropriate tool. The project can easily and naturally be extended in many ways.

1.5 OUTLINE OF THESIS

The PG/1 macro processor and the Filetab system are briefly described in Chapter 2 and Chapter 3 describes the development of the Filetab to COBOL program generator. The extension of macro processor facilities for implementation in the PG/2 version are described in Chapter 4.

In Chapter 5 the user interface and the development of a computer dominated dialogue with the user is discussed. Chapter 6 contains the specification for the Data base type self-tutorial system for the COBOL report program generator. The implementation of selected features of the system is described in Chapter 7. Finally, in Chapter 8 an appraisal of what has been achieved is made and pointers to areas which merit further study are identified.

The chapters are supported by material in a number of Appendices to which appropriate reference is made in the text. A separate folder of listing is also provided.
CHAPTER 2

BACKGROUND TO THE PRELIMINARY STUDY

2.1 INTRODUCTION

The purpose of the preliminary study was to establish the feasibility of generating effective COBOL report programs from a specification in an existing report generation language using a macro processor. The preliminary study was based on two existing pieces of software:

1. PG/1 Macro Processor [5].
2. NCC Filetab Package [7].

A subset of the Filetab language features was selected for use in specifying a request for a report. This enabled all the effort to be directed towards the development of macro definitions for generating the COBOL report program.

The following sections of this Chapter, supplemented by Appendices I and II, aim to provide background information about the software used during the preliminary study. Annotated examples of their use are also included.

2.2 PG/1 MACRO PROCESSOR

2.2.1 What is a Macro Processor?

A macro processor is described by Mandil [5] as 'a string transformation system which evaluates a source string, or macro call, to produce an object string'. The object string format is determined from text previously input as a macro definition, which consists of a macro name and a macro body containing the expansion string (Appendix I Section 3). Both the macro name and the macro body may contain one or more references to dummy parameters. A macro call consists of a previously defined macro name which may incorporate one or more actual parameters. When a macro call is
evaluated it is replaced by the expansion string with any actual call parameters replacing dummy parameters in the macro body (Appendix I Section 4).

2.2.2 Macro Processor features

The general features of a macro processor may be considered under five main headings:

1. Environment
2. Syntax
3. Evaluation of Macro calls
4. Macro-time facilities
5. Implementation methods

The environment in which a macro processor is embedded is determined by the language into which the macro processor maps i.e. its base language. A special-purpose macro processor is one that is applicable only to a single base language whereas a general-purpose macro processor is one that is applicable to a wide range of base languages.

The syntax of the macro processor governs the form of macro name, formal parameters, replacement body and macro calls. The syntax of the latter is of most importance because it influences the convenience of use and the range of applications of the macro processor.

There are two ways in which macro calls, and in particular nested macro calls, can be evaluated. In 'call by value', the nested call is evaluated immediately it is encountered and replaced by its value. In 'call by name', the outer call is fully expanded first.

The macro-time facilities constitute the programming language of the macro processor. They are mainly instructions that define and manipulate the macro-time entities and occur as
statements within the macro body.

Three considerations are fundamental to the implementation methods of a macro processor:

1. The number of times the source text is scanned.
2. The way in which macro-time statements are handled.
3. The storage of text by means of a stack or the use of a list structure.

2.2.3 Principal features of PG/1

The PG/1 general-purpose macro processor is a successor to the MP/1 macro processor [6], which has the following properties:

1. Macro calls are acceptable in certain contexts only and macro name recognition is then carried out character by character.
2. Nested macro calls are evaluated immediately they are encountered.
3. Nested macro definitions and definitions within macro calls are not permitted.
4. Macro-time statements are pre-compiled.
5. Text storage is in a stack-based structure.
6. One pass is made over the source text.
7. Special warning markers are employed.

The MP/1 macro processor was biased towards FORTRAN, but the enhancements provided in the PG/1 version by means of macro-time facilities tended to remove the bias and make PG/1 suitable for the generation of COBOL. The new macro-time facilities provided by the PG/1 macro processor include:

1. The ability to write messages to the user, read messages input by the user, construct output messages and analyse input messages.
2. Symbol and character manipulation. This involves the creation of a few macro-time stacks and associated macro-time statements.

3. The creation of facilities which allow the repetitive use of a group of macro-time statements with a provision for them to be dynamically parametized.

2.2.4 Minor modifications to the PG/1 macro processor

At the outset of this project it became apparent that three minor modifications to the existing PG/1 processor would be advantageous for the purposes of generating COBOL statements from Filetab directives.

1. Additional long string variables would be required so the number available was increased from three to ten.

2. The maximum length of a string variable should be increased from 60 to 72 characters so that it could store all the characters in either the input or output buffer.

3. The FORTRAN convention of character six in a statement as a continuation marker for macro-time statements requiring more than one line was suspended, because it interfered with the generation of statements containing characters in the sixth position.

The above modifications were therefore incorporated in the version of the PG/1 macro processor used for the preliminary investigation, whose facilities are described in Appendix I and illustrated in the following subsection.
Example 1.

This example shows the statements for filing the definition of macro FILECHECK in the TESTRUN subfile and then executing the FILECHECK macro. The macro itself requests input from the user and generates two output records which are appended to the COBOLPROG subfile.

```plaintext
%MACRO
%BEGIN

FILE,0,TESTRUN

%DEF FILECHECK

1 READ

CALLFSTR(LV1,1,5) IF LV1.EQ.'FILE',GOTO 2

CALLCOPY(1,INVALID*FILE*RECORD)

WRITE
```

Initialise the PG/1 system (Appendix I Section 6.1 and 6.2)

Cause all subsequent statements up to but excluding the one containing the 8 % characters to be stored at the beginning of the TESTRUN subfile. (Appendix I Section 6.5)

Define the macro named FILECHECK. The following statements up to %END form the macro body. (Appendix I Section 3)

Read a record into the PG/1 input buffer. (Appendix I Section 7.17)

Extract the first five characters of the input buffer into local variable LV1. (Appendix I Section 7.4)

Test if the contents of LV1 is equal to 'FILE and if so pass control to the statement labelled 2. (Appendix I Section 5)

Set up error message in the first 20 characters of the PG/1 output buffer. (Appendix I Section 7.3)

Write out the error message from the output buffer. (Appendix I Section 7.24)
% GOTO 1

%2 CALLFSTR(&LV2,6,8)
Pass control back to the statement with label 1.
(Appendix I Section 7.11)

% CALLFSTR(&LV3,14,4)
Extract 8 characters from the input buffer into local variable &LV2
starting at position 6 of the buffer.
(Appendix I Section 7.4)

FD TESTFILE LABEL RECORD STANDARD
VALUE OF ID "&LV2&LV3"
Generate two expansion time statements.
(Appendix I Section 5)

%END
End of macro definition
(Appendix I Section 3)

% LOAD,TESTRUN
Load and validate the contents of the TESTRUN subfile which contains the macro FILECHECK.
(Appendix I Section 6.7)

% FILECHECK
Call the FILECHECK macro
(Appendix I Section 4)

*FILEPAYROLLDATA
Request input record from the user.

%END
% SAVE,1,COBOLPROG
Terminate evaluation and append the generated statements to the COBOLPROG subfile.
(Appendix I Section 6.3 and 6.11)

% FINISH
Terminate PG/1 macro processor run.
(Appendix I Section 6.4)

When expanded the statements generated and appended to the COBOLPROG subfile will be as follows:

FD TESTFILE LABEL RECORD STANDARD
VALUE OF ID "PAYROLLDATA "

FD TESTFILE LABEL RECORD STANDARD
VALUE OF ID "&LV2&LV3"
Generate two expansion time statements.
(Appendix I Section 5)

%END
End of macro definition
(Appendix I Section 3)

% LOAD,TESTRUN
Load and validate the contents of the TESTRUN subfile which contains the macro FILECHECK.
(Appendix I Section 6.7)

% FILECHECK
Call the FILECHECK macro
(Appendix I Section 4)

*FILEPAYROLLDATA
Request input record from the user.

%END
% SAVE,1,COBOLPROG
Terminate evaluation and append the generated statements to the COBOLPROG subfile.
(Appendix I Section 6.3 and 6.11)

% FINISH
Terminate PG/1 macro processor run.
(Appendix I Section 6.4)

When expanded the statements generated and appended to the COBOLPROG subfile will be as follows:

FD TESTFILE LABEL RECORD STANDARD
VALUE OF ID "PAYROLLDATA "

FD TESTFILE LABEL RECORD STANDARD
VALUE OF ID "&LV2&LV3"
Generate two expansion time statements.
(Appendix I Section 5)

%END
End of macro definition
(Appendix I Section 3)

% LOAD,TESTRUN
Load and validate the contents of the TESTRUN subfile which contains the macro FILECHECK.
(Appendix I Section 6.7)

% FILECHECK
Call the FILECHECK macro
(Appendix I Section 4)

*FILEPAYROLLDATA
Request input record from the user.

%END
% SAVE,1,COBOLPROG
Terminate evaluation and append the generated statements to the COBOLPROG subfile.
(Appendix I Section 6.3 and 6.11)

% FINISH
Terminate PG/1 macro processor run.
(Appendix I Section 6.4)
Example 2.

This example shows the statements for defining and executing the INSERT macro which inserts the character supplied as an argument into three generated expansion-time statements.

```
%MACRO.
%BEGIN
%DEF INSERT©©
©1@FILE,0,GROWFILE©©
©1©DEF GROWMACRO
©1© LABELOFF
%END
% INSERT%
% END
%PRINT
%FINISH
```

When printed the generated statements will be as follows:

```
%FILE,0,GROWFILE©©
%DEF GROWMACRO
% LABELOFF
```

PG/1 puts the items generated into a buffer until the run terminates.
2.3 FILETAB

2.3.1 Introduction

Report writer programs are designed to print out an analysis of data located on one or more files. The data will usually be in some sequence of keys and the report will consist of totals or analyses performed on various groups of data records, each analysis or total being produced when a key change occurs. As input to the report program the user specifies the format of the files concerned, the details of the format and contents of the printed report and any rules for creating totals etc. Report writer programs are usually applied to relatively long runs of relatively straightforward data processing.

2.3.2 Features of Filetab

The NCC Filetab system is a general purpose tabulator and report printing system. It is designed to be such that non-technical staff with no programming knowledge may specify the required report.

All user requirements including the format of output records are specified by means of directive and parameter records. A single character is used to identify the way in which a data field is processed by the system and also serves to identify the contents of the data field in subsequent parameters. The Filetab system is essentially an interpreter in that it reads the parameter records, translates them into an appropriate internal language and then executes the internal program one step at a time.

The Filetab system accepts data from input files on cards, magnetic tape and disc for serial processing. Only the fields in the input data files required in the output need be identified to the system. The features available permit the user to accumulate and print decimal, binary and sterling fields and
information from any field may be transferred to the output report.

The parameters defining the report allow the user to specify a simple list of one or more print lines per record with totals at control changes. Alternatively a 'totals only' tabulation of printing when control changes occur may be requested. The page layout may include a report title, page heading and control break headings. The page number, current date and time, descriptive literals and any input field may appear in an output line or heading. Where desired fields may be edited.

The full Filetab facilities are very comprehensive but the program may be used at two levels of complexity. The lower level permits a complete data file to be processed in order to produce a report. At the higher level, records may be selected, additional processing specified and subsidiary files input and/or created. This higher level processing is specified in the form of decision table directives which may be entered at five so called decision points in the processing, e.g. immediately after reading a record from the main input data file, before printing a line of output.

The Filetab directives are written in free format and may contain special items of information called parameters which are separated by visible spaces. Many of the operands are optional and some directives may be omitted from the set. Where operands and directives are omitted, previously defined default values are assumed.
2.3.3 Selected features

For the purpose of this study a subset of the lower level Filetab facilities were selected for implementation, sufficient to provide the user with a simple flexible and useful report writing tool.

Of the Filetab features available only those concerned with the specification of a Main data file and the simpler options of the Report format specification were considered. The Filetab directive statements for these limited features provide adequate information from which a complete COBOL report program can be generated. Although the use of auxiliary Input and Output files, Data Selection and/or Editing and Miscellaneous Filetab options were excluded from the preliminary study investigations, some are covered in the later part of the project (Chapter 5).

Filetab directives may have either the * or the / symbol as the first character, but, for this study, only * was permitted.

Options from the following Filetab directives were selected for implementation:

*FILE (Appendix II Section 3.1)
*INL (Appendix II Section 3.2)
*TITLE (Appendix II Section 3.3)
*HEAD (Appendix II Section 3.4)
*OUT (Appendix II Section 3.5)
*comment (Appendix II Section 3.6)
*GO (Appendix II Section 3.7)

A input sequence for Filetab directives is laid down but, as in this study error handling was minimal, the input order was relaxed save that the *GO directive had to be the last directive for a run.
At this stage in the project program of work only the translation process itself was of interest, so the investigation was limited to see how valid statements could be used to generate a COBOL program. Error checking and the handling of invalid user statements are studied later in the project.

In this study default values for some missing parameters in the Filetab directive were assumed, but no attempt to set default values for missing directives as a whole was made.

In Appendix II only those Filetab language features selected for implementation in this study are described. The number and ranges of parameters which may be used for specific purposes have in some cases been restricted below those normally available in Filetab. Full details of all Filetab facilities are to be found in the appropriate NCC manual [7].

2.3.4 A sample of Filetab

The way in which Filetab directives are used to specify a report is shown in the following example where it is desired to read, extract, total and print data from a file on magnetic tape named STORES-DATA.

The key field positions in each record of the file in major to minor key order occupy positions 0 to 1, 2 to 5 and 6 respectively. In addition to these items, the description data from positions 7 to 18 of each record is to be printed, while the quantity data which occupies positions 20 to 22 is to be both accumulated and printed. The record format illustrated below shows the field specification character assigned to each report field, shaded fields do not feature in the report.
### Stores List

<table>
<thead>
<tr>
<th>Class</th>
<th>Item Type</th>
<th>Quantity</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ITEM TYPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0020A SCREW 2.5CM</td>
<td>870</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0020D SCREW 3.0CM</td>
<td>204</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0020F SCREW 5.0CM</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUB-TOTAL</td>
<td>1170</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0030C BOLTS 10CM</td>
<td>440</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0030D BOLTS 15CM</td>
<td>129</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUB-TOTAL</td>
<td>569</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOTAL</td>
<td>1759</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Class

<table>
<thead>
<tr>
<th>Item Type</th>
<th>Quantity</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM TYPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0010A SAW LARGE</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0010B SAW MED</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0010S SAW SMALL</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUB-TOTAL</td>
<td>27</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.1** Report layout for Filetab sample
Figure 2.1 illustrates the layout of the desired report, which is to have a title and have the lines of the output listing separated by control break headings and totalling lines. The latter two types of output are printed when a sequence break (Appendix II Section 1) in the input data for a non-minor key field is detected. The totalling lines contain the accumulated totals for the data prior to the sequence break and the control break headings provide the headings for the data records following the sequence break. The notes at the right hand side of the figure indicate which report lines are specified by which Filetab directive.

The Filetab directives for producing the report are shown below:

*FILE MT STORES-DATA
*INL
 L 6/1, M 2/4, N 0/2, A 7/12, 1 20/3
*TITLE 2,1
STORES LIST
*HEAD M 1,1
 ITEM TYPE
*HEAD N 1,1
CLASS
*OUT L 1,1
 NN MMMML AAAAAAAAAAA 111
*OUT M 1,3
 'SUB-TOTAL' 1111111
*OUT N 2,2
 'TOTAL' 1111111
*GO
The *FILE directive specifies that the STORES-DATA file is on magnetic tape. The parameters of the *INL directive relate the field specifying characters to the actual position of the field within the record. All the fields in the example are specified using the start position and length information.

The *TITLE directive specifies the line spacing before and after the report title line. The text and position of the title within the print line are indicated by the contents of the following parameter record. The headings to be output for each new value of the M and N control fields are specified by the ♦HEAD M and ♦HEAD N directives. Their format and parameter records are similar to that used for the *TITLE directive. In this example both headings contain only text information.

The format for listing the input data is specified by the *OUT L directive and its following parameter record. Again the directive record specifies details of the line spacing before and after the print line. The field specifying characters in the parameter record indicate both the field to be printed and the actual positions it is to occupy. In this example the input data is listed without any additional text.

The *OUT M and *OUT N directives specify the format of totalling output to be printed when a change in the input value of the M and N control fields, respectively, are detected. The directive record indicates the line spacing required before and after the print line, whose format is specified in the following parameter record. The totalling field specifier in the parameter record indicates the print positions to be occupied by the accumulated total. In this example the totals at each control level are accompanied by text information. The latter is coded in the desired print positions and enclosed in quotation marks which do not appear in the printed report.
The end of the Filetab specification is indicated by the presence of the "GO directive."
CHAPTER 3

DEVELOPMENT OF A FILETAB TO COBOL PROGRAM GENERATOR

3.1 INTRODUCTION

This chapter, supplemented by the material in Appendix III, describes the development of a Filetab to COBOL program generator using the PG/1 macro processor. The description covers the following main topics:

1. The initial approach to the problem by the analysis of information contained in Filetab directives and its relation to the divisions of the COBOL program.

2. The problems of storing and sequencing the generated COBOL statements.

3. The adoption of a three phase approach for the design of the system to generate and synthesize a complete COBOL program.

4. The break-down of a Filetab directive into COBOL statements using the *INL directive as an example.

5. The macro processing of Filetab directives including the identification of the directive, the initialisation processing and an example of parameter processing based on the *INL directive.

6. An assessment of the Filetab to COBOL generator in terms of the quality of the program generated, the influence of the problem specifying language and the limitations of the macro processor.

The chapter ends with a summary, appraisal and conclusions.
Figure 3.1 Contributions by Filetab directives to COBOL program divisions
3.2 INITIAL APPROACH

The approach to the problem of generating a COBOL program from the information supplied by Filetab directives was influenced by two factors. First, the need to make best use of the PG/1 macro processor facilities, and secondly the desire to generate COBOL statements which could be structured into an efficient program.

Since COBOL programs are divided into four divisions, viz Identification, Environment, Data and Procedure, it was first necessary to analyse the various Filetab directives in order to see how the information they provide relates to the four COBOL divisions. As shown in Figure 3.1, each Filetab directive and its associated parameters provides information required to generate statements in two or more divisions of a COBOL program.

A more detailed analysis of Filetab statements revealed that the information from one directive can influence the contents of several paragraphs within the Procedure division. It is not therefore possible to generate the COBOL statements in the order in which they are required in the program. Some mechanism for storing and ordering the generated statements has to be found. The solution of these major problems greatly influenced the design of the Filetab to COBOL program generator.

3.3 STORING AND SEQUENCING GENERATED COBOL STATEMENTS

In this section, solutions to the problems of storing and sequencing the generated COBOL statements using the PG/1 macro processor facilities are discussed. Although the following four solutions were considered, only the last was regarded as being practical:

1. Use of SAVE-blocks.
2. Use of %SAVE and %LOAD.
3. The three phase system using data subfiles.
4. The three phase system using 'grown' macros.

3.3.1 Use of SAVE-blocks

The PG/1 macro-time statements SAVE, ENDSAVE and COPY were first considered as a means of providing temporary intermediate storage for the generated COBOL statements by the use of SAVE-blocks (Appendix I Sections 7.7, 7.9 and 7.21). They were, however, rejected because with the existing processor it is not possible to append data to that already in an existing SAVE-block. A new SAVE-block would therefore be required each time a generated COBOL statement was not destined to follow sequentially after its predecessor. This would lead to unwieldy processing in order to recover data from the many SAVE-blocks in the sequence required for the COBOL program.

Sequencing is not the only problem, for SAVE-blocks are part of the stack mechanism of the macro processor and therefore take up space which could be used for other purposes. With a verbose language like COBOL the contents of the SAVE-blocks would take up too much stack space.

3.3.2 Use of %SAVE and %LOAD

As the internal storage of generated statements proved impractical, attention was turned to the system macros %SAVE and %LOAD. They provide a means of access to and retrieval from a disc file storage system associated with the processor (Appendix I Sections 1, 6.7 and 6.11).

The %SAVE system macro causes the contents of the output stack to be stored in the referenced subfile. Depending on the value of a parameter storing starts at the beginning of the
Figure 3.2 The three phase system using data subfiles
subfile or immediately after the present subfile contents. However, system macros cannot appear between the %DEF and %END statements which delimit a macro definition. To use the %SAVE command directly would mean devising many small macro definitions and requiring the user to sequence the loading and calls to each by means of appropriate %LOAD and %SAVE commands.

e.g.

%LOAD,MACRO1
%MACRO1
%END
%SAVE,0,INPUTREC

Load and call macro MACRO1.

Save statements generated by MACRO1 at the beginning of the INPUTREC subfile.

%LOAD,MACRO2
%MACRO2
%END
%SAVE,0,ADDPARA

Load and call MACRO2.

Save statements generated by MACRO2 at the beginning of the ADDPARA subfile.

%LOAD,MACRO3
%MACRO3
%END
%SAVE,1,INPUTREC

Load and call macro MACRO3.

Append the statements generated by MACRO3 to those already in the INPUTREC subfile.

and so on ...

This approach was rejected as too cumbersome and error prone for even an experienced user (let alone the inexperienced user).

3.3.3 The three phase system using data subfiles

A solution to the sequencing problem can be reached by using the %FILE command and adopting a three phase approach for the generation of the complete COBOL program. The three phases illustrated in Figure 3.2 cover the following topics:

1. Statement generation.
2. Filing the generated statements.
3. Synthesis of the complete COBOL program.
The first phase requires that the macro definitions generating the COBOL statements should also generate their system filing commands. Thus each group of COBOL statements in the output stack would be contained between a %FILE command specifying the destination subfile and the associated terminator record of eight '%' characters (Appendix I Section 6.5). For example the statements generated in the macro processor output stack have the following form:

```
%FILE,0,INPUTREC
```

COBOL statements for defining record fields which are destined for the INPUTREC subfile.

```
%FILE,0,ADDPARA
```

COBOL statements for accumulating totals which are destined for the ADDPARA subfile.

```
%FILE,1,INPUTREC
```

Further COBOL statements for defining input record fields which are destined to be appended to the INPUTREC subfile.

and so on ...

The contents of the output stack is saved for use during the second phase.

During the second phase the generated statements are loaded back into the system so that the generated filing statements can be obeyed. The generated COBOL statements are thus filed in the specified subfiles.

The third phase is concerned with the synthesis of the complete COBOL program from a basic skeleton and the generated statements stored in the various subfiles. To retrieve the generated COBOL statements from the subfiles the %LOAD command must be used, but this is a system macro which cannot be used from within a macro definition. It is therefore necessary to intersperse %LOAD commands between many small macros in order to
Figure 3.3 The three phase system using 'grown' macros
generate the complete COBOL program.

e.g.

\%LOAD, PROGRAMPART1% PROGRAMPART1

Load and call the PROGRAMPART1 macro which generates the first part of the COBOL program.

\%LOAD, INPUTREC% INPUTREC

Load and include the COBOL statements stored in the INPUTREC subfile.

\%LOAD, PROGRAMPART2% PROGRAMPART2

Load and call the PROGRAMPART2 macro which generates the next part of the COBOL program.

\%LOAD, ADDPARA% ADDPARA

Load and include the COBOL statements stored in the ADDPARA subfile.

\%LOAD, PROGRAMPART3% PROGRAMPART3

and so on ....

Although the storage problem is overcome by this method, this approach to the sequencing problem is both tedious and error prone.

3.3.4 The three phase system using 'grown' macros

When a subfile containing data is loaded the PG/1 system transfers its contents to the output stack from where it cannot be accessed from within a macro definition. In contrast, if a subfile containing a macro definition is loaded, the macro is compiled and stored in the definition stack. Access to one macro from within another is possible by means of the PG/1 macro nesting facility (Appendix I Section 4).

By changing the subfiles used in the three phase approach outlined in Section 3.3.3 from data subfiles to simple macro definition subfiles, the macro nesting facility can be used to simplify the task of synthesizing the complete COBOL program. The three phase generation of a COBOL program from Filetab directives illustrated in Figure 3.3 may be summarized as follows:

1. The input of data in Filetab directive format and the generation of COBOL, system macro and macro-time statements.
2. The filing of the generated statements in the appropriate subfiles in order to 'grow' simple macro definitions.

3. The synthesis of the complete COBOL program using a macro definition which makes nested calls to the macros 'grown' during the previous phase.

When the data subfiles are changed to simple macro definitions the macro bodies consist largely of COBOL statements. In addition to the COBOL and system filing statements, the macro defining statements, %DEF and %END, and some simple macro-time statements are also generated during phase 1. The following outline illustrates the phase 1 contents of the output stack for growing the INPUTREC macro definition in the subfile of the same name:

```
%FILE,0,INPUTREC
%DEF  INPUTREC

%FILE,1,INPUTREC

%FILE,1,INPUTREC

%END
```

System macro statements defining macro INPUTREC and possibly some macro-time and COBOL statements.

More COBOL statements for inclusion in the body of the INPUTREC macro.

Some further COBOL and macro-time statements followed by the system macro statement for completing the INPUTREC macro.

and so on ...

During the second phase the generated statements are loaded back into the system so that the filing commands can be obeyed. The generated macro definition, macro-time and COBOL statements are thus filed in the specified macro definition subfile.

The task of synthesizing the complete COBOL program from the macros 'grown' during the first two phases is no longer
complicated from the user's point of view. It consists of loading the required macros and initiating a call to the macro definition which generates the program shell containing mainly static information. This macro, called GOPART2, automatically makes nested calls to the 'grown' macros whose contents depend entirely on the options specified in the original Filetab directives.

e.g.

%LOAD, INPUTREC, ADDPARA, ...., ...., $           Load 'grown' macros.

%LOAD, GOPART2$                                    Load and call the macro
%    GOPART2                                        which generates the shell of
                                                   the COBOL program.

The GOPART2 macro has the following form:

%DEF  GOPART2 Define program generating macro.
    Generate part of the COBOL program shell.

    [INPUTREC] Nested call to the 'grown' macro whose
               body contains the COBOL statements defining the input record format.

    Generate more of the COBOL program shell.

    [ADDPARA] Nested call to the 'grown' macro whose
               body contains the COBOL arithmetic processing statements.

%END                                                   End of macro definition.
<table>
<thead>
<tr>
<th>Macro name</th>
<th>Contents</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>PART1</td>
<td>Statements specifying the COBOL compiler options.</td>
<td>Steering Lines</td>
</tr>
<tr>
<td>CONSTANTS</td>
<td>Macro-time statements restoring the values of certain global and string variables to those set during the first phase of the generating system.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>INPUTREC</td>
<td>Input file data record descriptions.</td>
<td></td>
</tr>
<tr>
<td>CTRLBRKE</td>
<td>Data descriptions for each control field used in the report.</td>
<td></td>
</tr>
<tr>
<td>TITLES</td>
<td>Data descriptions for the report title records.</td>
<td>Data Division</td>
</tr>
<tr>
<td>HEADINGS</td>
<td>Data descriptions for the control break heading records.</td>
<td></td>
</tr>
<tr>
<td>OUTREC</td>
<td>Data descriptions for detail and control total lines.</td>
<td></td>
</tr>
<tr>
<td>SAVETOTL</td>
<td>Data descriptions for each totalling field used in the report.</td>
<td></td>
</tr>
<tr>
<td>NOTEPARA</td>
<td>COBOL NOTE statement containing Filetab comment records.</td>
<td></td>
</tr>
<tr>
<td>ADDPARA</td>
<td>Statements to add totalling fields in the input record to the corresponding total field at the lowest control level.</td>
<td></td>
</tr>
<tr>
<td>TITLEPAR</td>
<td>Write statements to output the report title lines.</td>
<td></td>
</tr>
<tr>
<td>PAGEPARA</td>
<td>Statements which initiate the writing of control group headings.</td>
<td>Procedure Division</td>
</tr>
<tr>
<td>MOVEPARA</td>
<td>Statements to move fields from the input record to the detail line records.</td>
<td></td>
</tr>
<tr>
<td>WRITEPARA</td>
<td>Statements which detect the need for a new page and write out the detail lines for the current input record.</td>
<td></td>
</tr>
<tr>
<td>BREAKPARA</td>
<td>Paragraphs containing statements to detect and process a sequence break in the input record control fields.</td>
<td></td>
</tr>
</tbody>
</table>

continued
3.4 THE DESIGN OF THE COBOL PROGRAM GENERATOR

In this section further consideration is given to the design of the Filetab to COBOL program generator in terms of the three phases outlined in Section 3.3.4.

3.4.1 Phase 1 - Statement generation

A detailed analysis of the data contained in the Filetab directives showed that there were fifteen groups into which the generated statements would fall. Six were associated with the Data division of the program, while the remaining nine were associated with the Procedure division. Each group of generated statements would therefore need to be 'grown' into a separate macro definition.

In addition two further macros are required, one to pass global and string variables between phases 1 and 3, and another to contain Steering Lines data. Steering Lines are parameters which specify to an ICL 1900 series COBOL compiler the options required. Table 3.1 shows the name and contents of each of the seventeen macro definitions whose statements are generated during the first phase of the COBOL generating system.

Figure 3.4 shows to which macro definitions each Filetab directive may contribute and how, in turn, the former contribute to the various Divisions and Sections of the COBOL program. For example, the following *INL directive contains information which affects the statements generated for the input record description (INPUTREC macro), the data description for totalling fields (SAVETOTL macro), the data descriptions for the sequence control fields (CTRLBRKE macro) and the processing of totalling fields (ADDPARA macro).
Table 3.1 continued

<table>
<thead>
<tr>
<th>Macro name</th>
<th>Contents</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINALPAR</td>
<td>Statements which initiate the writing of the final control total lines at the end of the report.</td>
<td>Procedure Division</td>
</tr>
<tr>
<td>OTHERPAR</td>
<td>Paragraphs for testing if a new control heading is required, writing control heading and control total lines.</td>
<td></td>
</tr>
</tbody>
</table>
Figure 3.4 Filetab directive contributions to 'grown' macros and the generated COBOL program
The number of macro definitions which are influenced by certain directives depends on the parameters specified in the directive by the user. If, for example, the totalling field was omitted from the above *INL directive, COBOL statements would not be generated for the SAVETOTL and ADDPARA macros.

3.4.2 Phase 2 - Forms of the 'grown' macros

In order to avoid storing the Filetab directives only one pass is made over the input statements during phase 1 of the generating system. This means that the bodies of the macros 'grown' during the second phase of the system take one of the following forms:

1. Contain no generated COBOL statements at all.
2. Contain simple Procedure division paragraphs consisting of an EXIT statement.
3. Contain the whole or part of a Procedure division paragraph which defines some processing.
4. Contain Data division statements which define input or output records or Working Storage items.
5. Contain the Steering Lines necessary to compile the generated COBOL program on an ICL 1900 series computer.
6. Contain macro-time statements for setting up global and string constants which are needed in the third phase of the generating system.
3.4.3 Phase 3 - skeleton COBOL program

The main aim when designing the skeleton program was to keep the logical flow simple, but at the same time use structures which can accommodate the many variations likely to be specified by the Filetab input data. To this end the Procedure division of the program was largely built up from a series of PERFORM statements. These PERFORM statements refer to paragraphs wholly or partly contained in the macros 'grown' during the first two phases of the system. This approach is similar to the extensive use of parameterized subroutines in the FORTRAN program generator (3.8).

In some instances a PERFORM statement is only generated if a global variable has previously been set to indicate that the required statements have been generated (3.4.2 Case 1). In other instances it is possible to unconditionally include a PERFORM statement in the generated program, because the paragraph referenced contains the generated statements to carry out the processing, or it contains an EXIT statement if no processing is required (3.4.2 Cases 2 and 3).

The outline flowchart for the shell of the Procedure division of the generated COBOL program is illustrated in Figure 3.5. The listing of the GOPART2 macro, which generates the complete COBOL program by making nested calls to the 'grown' macros, is included in the separate folder (Item 3).

In the Data division, Working Storage section of the generated COBOL program additional FILLER fields are included in certain group data descriptions. They are present in order to avoid compilation diagnostics when the 'grown' macro contains no generated statements for the lower levels of the group. For example, when there are no control totals to be accumulated the group description will appear as follows:
Figure 3.5 Outline of processing in the generated COBOL program

START

OPEN FILES

INITIALISATION

READ INPUT RECORD

SET UP CONTROL BREAK DATA

ACCUMULATION OF TOTALS IF REQUIRED

REPORT TITLE PROCESSING

PAGE HEADING PROCESSING

MOVEMENT OF DATA TO OUTPUT RECORD STORAGE

OUTPUT OF DATA RECORDS TO REPORT FILE

READ INPUT RECORD AT END

CONTROL BREAK PROCESSING

ACCUMULATION OF TOTALS, IF REQUIRED

FINAL PROCESSING

CLOSE FILES

STOP RUN
The WRITE FROM option of the WRITE verb is used in the generated program so that literals in the various record types can be set up once and for all. Only variable data need then be moved into position prior to writing out the record.

3.5 EXAMPLE OF THE BREAK-DOWN OF A FILETAB DIRECTIVE INTO COBOL STATEMENTS

By taking the *INL filetab directive as an example this section serves to show how the elements in a parameter are translated into COBOL. It also emphasises the volume and variety of COBOL statements which may be generated from the information contained in a single Filetab directive. Notes describing the generation of COBOL statements from the other Filetab directives implemented in this study are given in Appendix III.

An *INLIST directive defining two control level fields, a transfer and a totalling field of the form shown below will require the generation of COBOL statements for inclusion in four macros, namely INPUTREC, CTRLBKE, SAVETOTL and ADDPARA. Consider for example:

*INL
L 4/2, M 0/4, A 7/10, 1 19/3

The fields in a COBOL record description are described in start position order. In contrast Filetab does not require the fields to be specified in the order in which they appear in the input record. Both Filetab and COBOL allow fields to be redefined and such fields may overlap others. The greater flexibility in Filetab is overcome by making each Filetab field generate a complete new record description which redefines the previous one. Thus each COBOL record description appears to contain only one named field with the unused positions in the record described by
Magnetic tape files used on an ICL 1900 computer contain a sentinel record which, among other details, includes the maximum record length used in the file. This information is used by the COBOL housekeeping routines and overrules that specified in the COBOL program. Thus, to prevent execution errors caused by a conflict between sentinel and program data, a standard maximum input record length of 2048 characters was adopted. This figure is in line with the ICL conventions for COBOL programs of this size and type using magnetic tape files.

Since COBOL data-names cannot be entirely numeric the Filetab totalling field specifiers are prefixed by the letter A when COBOL program statements to process them are generated. Other types of field specifiers remain unchanged when used as COBOL data-names.

The lowest level control field, L, only contributes to the Data division input record specifications, which are filed in the INPUTREC macro. It causes the following statements to be generated:

```
02 FILLER-1 REDEFINES INREC.
03 FILLER PICTURE X(4).
03 L PICTURE X(2).
03 FILLER PICTURE X(2).
03 FILLER PICTURE X(120) OCCURS 17.
```

The ICL 1900 COBOL compiler imposes a maximum size of 120 characters on an alphanumeric field, hence the need to use the OCCURS clause and have two FILLER statements to fill out the record to the correct length.

The M field, being a control field not of the lowest level, contributes COBOL statements to both the INPUTREC and CTRLBROKE macros. The latter macro contains statements used to build up a group structure in the Data division, which is used by the Procedure division to detect a sequence break in the control
fields.

Thus control field M will cause the following statements to be generated for inclusion in the INPUTREC macro:

```
02 FILLER-2 REDEFINES FILLER-1.
 03 M PICTURE X(4).
 03 FILLER PICTURE X(4).
 03 FILLER PICTURE X(120) OCCURS 17.
```

Only one statement is generated by this control field for inclusion in the CTRLBRKE macro and this is as follows:

```
02 M PICTURE X(4).
```

Transfer field A is similar to the lowest control level field in that it only causes statements to be generated for inclusion in the INPUTREC macro.

Viz,

```
02 FILLER-3 REDEFINES FILLER-2.
 03 FILLER PICTURE X(7).
 03 A PICTURE X(10).
 03 FILLER PICTURE X(111).
 03 FILLER PICTURE X(120) OCCURS 16.
```

The totalling field, 1, contributes generated statements to three macros, namely INPUTREC, SAVETOTL and ADDPARA. The SAVETOTL macro is used to store statements required by the Data division for defining totalling fields for each control level of the report, except the minor control level. A Procedure division statement for the accumulation of the control total is required and this is generated for inclusion in a macro called ADDPARA. Thus the Filetab field specification 1 19/3 contributes the following statements to the macros indicated below:

INPUTREC macro:

```
02 FILLER-4 REDEFINES FILLER-3.
 03 FILLER PICTURE X(19).
 03 A1 PICTURE 9(3).
 03 FILLER PICTURE X(106).
 03 FILLER PICTURE X(120) OCCURS 16.
```

SAVETOTL macro:

```
03 A1 PICTURE 9(15) COMPUTATIONAL VALUE ZERO.
```
ADD PARA macro:

ADD A1 IN INREC TO A1 IN M-LEVEL.

While the extensive use of the COBOL REDEFINES facility clearly affects the program compilation time, its effect on program execution time is minimal.

3.6 THE MACRO PROCESSING OF FILETAB DIRECTIVES

The design of COBOLGEN, the macro for processing the Filetab directives and generating the statements which, in phase 2, are filed to 'grow' macros, is now considered. The following topics are discussed:

1. Outline processing for the identification of Filetab directives.

2. Initialisation processing.

3. Outline processing of Filetab directive parameters.

3.6.1 Outline processing for the identification of Filetab parameters.

As shown in Figure 3.6, a modular approach was used for the design of the COBOLGEN macro definition, which generates the COBOL statements from the Filetab directives.

The Initialisation section (3.6.2) is followed by a READ operation which inputs a Filetab directive. In the next section the directive, which may be one of seven valid types, is identified. Control then passes to the section where the particular directive and its associated parameter records, if any, are processed.

The processing of the parameter records continues within each section until a new directive record, starting with an * character, is detected. Control then returns to the section which identifies the new directive.
Figure 3.6 Outline processing for Phase 1 of the Filetab to COBOL program generator

START

INITIALISATION

READ FILETAB DIRECTIVE

*comment

Y

PROCESS COMMENT

N

*FILE

Y

PROCESS *FILE DIRECTIVE

N

*INL

Y

PROCESS *INL DIRECTIVE

N

*TITLE

Y

PROCESS *TITLE DIRECTIVE

N

*HEAD

Y

PROCESS *HEAD DIRECTIVE

N

*OUT

Y

PROCESS *OUT DIRECTIVE

N

*GO

Y

PROCESS *GO DIRECTIVE

N

ERROR CONDITION MESSAGE

FINISH
The exceptions to the above are the * comment and *GO directives. In the case of comments they are processed one at a time with control returning to the first READ operation. When the *GO directive is detected the outstanding COBOL and macro definition statements are generated, thus completing the first phase of the generating system.

A listing of the macro-time statements for the identification of Filetab directives is included in the separate folder (Item 1).

3.6.2 Initialisation processing

In order to 'grow' macro definitions it is necessary to be able to use the % ' and # characters in some of the generated statements. However, these characters have special meanings within the macro processor and are, respectively, the macro warning character, the literal delimiter and the label marker. Special processing is required in order to achieve their presence in the generated output. A common method for handling these special characters is not possible because of their different special meanings to the PG/1 macro processor.

The PG/1 macro processor stores macro definitions in core in an intermediate code which is executed when the macro is called. Text to be output by a macro definition is intersperced with the macro-time statements. In the execution phase a % character in the intermediate code indicates a call to a subroutine, hence the likelihood of confusion and error when the output text also contains % characters. To overcome this difficulty the % character is input to the macro definition in the form of an argument. Thus the COBOLGEN macro must be called by the following
During the development of the COBOLGEN macro it was necessary to depart from the default value for the argument marker as the @ character was an operating system control character on the host computer. Instead the / character was defined as the macro argument marker using the %MACRO system macro (Appendix I Section 6.1). Wherever a % character is required in the expansion-time statements of the COBOLGEN macro the macro argument /1/ is used.

The quote character which is used by the macro processor to delimit alphanumeric literals also presents a problem when it is required in a statement to be generated. This problem was overcome by reading the quote character from an input record and storing it in a global variable called %GQUO. The record containing the quote character must follow immediately after the call to the COBOLGEN macro and precede the Filetab records in the input stream.

The # character is set up as an alphanumeric literal in global variable #GHATC. The name of this variable is specially chosen to have the full four characters after the G so that it can be placed immediately in front of a string of characters in an expansion-time statement. A shorter name causes the macro processor to interpret the beginning of the character string as part of the global variable name.

Statements destined for some of the macros to be 'grown' may be generated at various points within the COBOLGEN macro, depending on the Filetab directives submitted as input. In order to handle this situation three small routines were devised to generate the statements for filing and defining the macros to be 'grown'.
The first routine generates the \%FILE and \%DEF statements for the subfile whose name is contained in global variable ^FIL. A LABELOFF macro-time statement is also generated so that the first six characters of any generated statement which follows are not ignored (Appendix I Section 7.13).

\%101 CONTINUE : INITIALISE  
/1/FILE,0,^FIL/  
/1/DEF ^FIL  
/1/ LABELOFF  
/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1/  
\%  
\% RETURN  

The second routine generates the system macro statement necessary to append subsequently generated statements to an existing macro definition subfile. The subfile name is again contained in global variable ^FIL.

\%102 CONTINUE : APPEND  
/1/FILE,1,^FIL/  
\% RETURN  

The last of the basic service routines generates the filing statements necessary to append the final \%END statement to the subfile containing the macro definition specified by the ^FIL global variable. It also generates the LABELON macro-time statement to negate the effect of the LABELOFF statement at the beginning of the 'grown' macro.

\%104 CONTINUE : FINALISE  
/1/FILE,1,^FIL/  
/1/ LABELON  
/1/END  
/1/1/1/1/1/1/1/1/1/1/1/  
\% RETURN  

\%
In addition to the processing to handle the special characters, the Initialisation section of the COBOLGEN macro causes the statements which initialise all the 'grown' macros to be generated. This latter processing consists of pairs of macro-time statements which set the name of the macro in global variable $GPIL and cause INITIALISE, the first of the above service routines, to be obeyed. Some of the 'grown' macro definitions have an associated global variable which is used to indicate whether or not the body of the 'grown' macro contains any generated COBOL statements (3.4.2 Case 1). These global variable indicators are set equal to zero during the Initialisation section of the COBOLGEN macro.

A listing of the Initialisation section of the COBOLGEN macro is included in the separate folder (Item 2).

3.6.3 Outline processing of Filetab directive parameters

The processing of Filetab directive parameters is illustrated by taking one example, namely the parameters of the *INL directive which was discussed in Section 3.5.

The flowchart in Figure 3.7 shows the outline processing for the Filetab *INL parameter records. Depending on the parameters data, this processing will generate COBOL, macro-time and system filing statements for one or more of the subfiles INPUTREC, SAVETO SAVETOTL, ADDPARA and CTRLBRKE (3.5). The APPEND service routine (3.6.2) is used to generate the filing commands which direct the generated COBOL statements to the desired subfile. It will be noted from Figure 3.7 that there are multiple call to append generated COBOL statements to the INPUTREC subfile and that these are interspersed with calls to append statements to other subfiles. This illustrates the convenience of the sequencing
Figure 3.7 Outline processing of Filetab *INL parameters

ENTRY

ZEROISE FIELD COUNTER

READ PARAMETER RECORD

% IN FIRST POSITION

EXIT

Determine length of parameter data

ADD 1 TO FIELD COUNTER

EXTRACT FIELD DESCRIPTION DETAILS

IF NECESSARY, CALCULATE FIELD LENGTH

INPUTREC GENERATE RECORD DESCRIPTION

INPUTREC GENERATE FILLER FIELD DESCRIPTIONS, IF REQUIRED

TOTALLING FIELD

SAVE TOTAL Accumulated total data description

CONTROL FIELD

INPUTREC GENERATE CONTROL DATA DESCRIPTION

INPUTREC GENERATE ALPHANUMERIC FIELD DESCRIPTION

ADDPARA GENERATE ADD STATEMENT

INPUTREC GENERATE FILLER FIELD DESCRIPTIONS, IF REQUIRED

END OF PARAMETER RECORD

INPUTREC GENERATE FILLER FIELD DESCRIPTIONS, IF REQUIRED
method adopted for the COBOL generating system (3.3.4).

In order to deal with the problem of overlapping fields and fields not specified in start position order, each Filetab field is made to generate a complete new record description which redefines the previous description (3.5). A count of input fields is maintained and is used in the nomenclature of the record descriptions when they are generated (Appendix III Section 4).

Parameter records are read and processed one at a time until a new Filetab directive record with an asterisk in the first position is detected. Control then passes to the macro-time statements which identify the directive (3.6.1).

The parameter record in the input buffer is squashed to remove imbedded spaces in order to simplify data extraction and to determine its length.

The field counter is incremented and the details for a field description are extracted (Appendix II Section 3.2.2). If the end position rather than field length was specified the latter is calculated for use when the COBOL field description is generated.

The COBOL statement defining the new record description is generated for filing in the INPUTREC subfile. Unless the current fields starts in the first position of the record, FILLER field description statements for the initial part of the record are generated. These too will be filed in the INPUTREC subfile.

If the current field is a totalling field a data description statement for the accumulated total is generated for filing in the SAVETOTL subfile. The input record numeric field description is then generated for filing in the INPUTREC subfile. Finally the ADD statement, which accumulates the total, is generated for inclusion in the ADDPARA subfile.
If the current field is a control field a control break data description is generated for inclusion in the CTRLBNKE subfile. Both control and transfer fields have an alphanumeric input record field description generated for filing in the INPUTREC subfile.

Unless the current field, regardless of type, ends in the last position of the record, FILLER field description statements are generated for the final part of the record. These will be filed in the INPUTREC subfile.

If, when the current field description has been processed, the input buffer data has been exhausted another parameter record is read. Otherwise the details for the next field description are processed.

Further details of the parameter processing, statements generated and macro-time variables used are given in Appendix III Section 3. A listing of the macro-time statements for processing *INL parameters is included in the separate folder (Item 5).

3.7 ASSESSMENT OF THE FILETAB TO COBOL PROGRAM GENERATOR

The processing described earlier in this chapter and that outlined in Appendix III demonstrates the feasibility of using the PG/1 macro processor to generate a complete syntactically correct COBOL program from a set of Filetab directives. At this stage we are not concerned with the full syntax checking of Filetab directives nor the semantics of the Filetab request. Some assessment of program size and efficiency, the elegance of the generated code, the influence of the problem specifying language and the limitations of the macro processor must, however, be made before the generation of COBOL programs by macro processor can be deemed a viable proposition.
3.7.1 Program size and efficiency

Some measure of the efficiency of the generated program can be assessed by comparing its run time statistics with those of a Filetab run on the same computer, using the same input data file with the same directive statements and producing an identical output report.

In a typical comparison exercise the following Filetab directives were used to generate a COBOL program:

* THIS IS A COMMENT
*TITLE 1,3
REPORT TITLE
TABN TO COBOL TEST
*FILE MT STUD-MAIN73
*HEAD M 1,1
CONTROL LEVEL M HEADING FIELD-1 FIELD-2 FIELD-3
*HEAD N 1,1
CONTROL LEVEL N HEADING
*INL
  L 7/1, M 6/1, N 4/2, A 0/4, B 4/4, 1 8/3
*OUT L 1,1
  AAAA     BBBB   111
*OUT M 1,1
  'SUB-TOTAL'  1111111
*OUT N 1,1
  'TOTAL'     1111111
*GO

TABN, the ICL 1900 series version of the Filetab program, was allocated 19K words of storage on an ICL 1907 computer and executed using the above directives. The data file contained 248 variable length records, the maximum size of which was 2048 characters. One second of Central Processor Unit (CPU) time was required to run the program, which used 296 write operations to print the report, list the directive statements and print the job statistics. In contrast TABC, the generated COBOL program consisting of 232 source records, required only 3½K words of storage on the same computer, but used two seconds of CPU for execution. The number of write operations required to produce an identical report and to print the job statistics rose to 557.
The significantly larger number of write operations required by the COBOL program is due to the inefficient method used to achieve the required spacing after any print group. The wasteful technique of printing a blank line after each print group needs to be replaced by the setting of a counter, which can be added to the ADVANCING count for the next WRITE statement. The process of printing the blank lines is a contributory factor in the higher usage of CPU time by the COBOL program. This latter statistic was, however, only available rounded to the nearest second and so was not a very sensitive measure of program efficiency.

The core requirements of the generated COBOL program compares favourably with those which would be required by a hand coded program, although the coding of the latter would be more elegant. Apart from the wasteful printing of blank lines, there are other areas where improvements could be made in the generated COBOL program. These would both improve the run time efficiency and reduce the compilation time by the use of fewer, shorter source language statements.

3.7.2 The elegance of the generated COBOL code

The program listing forms an important part of the documentation for a program. In the case of a generated program the listings of the input parameters and the generated source code are often the only pieces of documentation available. In such cases the elegance of the generated code is a contributory factor in the clarity of the program documentation.

The generation of the Data division input file record description is one area where an improvement could be made. The number of redefinitions could be reduced, if some attempt was made to relate each data field to the one that preceded it on the *INL parameter record. Thus, only when data fields overlapped, or were
not specified in start position order, would there be a need for a REDEFINES statement. Another opportunity for improvement occurs in the Procedure division of the generated program where, by sectionlising the paragraphs, a statement of the form:

PERFORM paragraph-name-1 THRU paragraph-name-2.

could be replaced by a statement of the form:

PERFORM section-name.

Although not shown up by the particular Filetab example quoted above, a deficiency exists in the method used by the COBOL program to detect the need for a new page when printing a group of output lines. The "HEAD L directive is used to specify, either explicitly or by default, the maximum number of lines per report page (Appendix II Section 3.4). The COBOL program, however, only tests the line count value prior to printing a group of output lines. Thus, if an output group consists of several lines, the page may turn out to have slightly more than the permitted maximum. A more sophisticated method for end of page detection is required to overcome this situation. Also it must ensure the suppression of control group headings near the bottom of a page when there will not be enough room for at least one detail line to follow. These features are taken in to account in the design of the more elaborate COBOL report program generating system described in Chapters 6 and 7.

3.7.3 The influence of the problem specifying language

The use of the pre-defined Filetab language to specify the problem to the macro processor, so that it, in turn, could generate a COBOL program to provide a solution, proved valuable. It not only established the feasibility of the approach, but also served to highlight areas where it was more difficult to
translate the problem language into COBOL statements. For example, one such difficulty was caused by the fact that both Filetab and the macro processor used the single quotation mark character (') to delimit a text literal. Special processing, as outlined in Section 3.6.2, was needed to overcome the problem. Thus, in the design of any problem specifying dialogue for use with this macro processor, the choice of a different text literal would be advisable. As the dialect of COBOL being generated by the system uses the double quotation mark character ("') to delimit alphanumeric literals, it would be appropriate to adopt the same character, for the same purpose, in the design of a problem specifying dialogue.

The Filetab *FILE directive has no parameter which describes the maximum record length of the file, hence the adoption of 2048 characters as the record size by the COBOL generating macros (3.5). If this piece of data could be supplied in a computer/user dialogue, the generated program would require less core storage and have fewer FILLER statements when the file's record length was less than the 2048 assumed at present.

The experience gained from Filetab to COBOL program translation will influence two other aspects in the design of a computer/user dialogue for the generation of a COBOL program.

The first is related to the design of the format of report lines. The parameter records for the *TITLE, *HEAD and *OUT Filetab directives provide a simple straightforward way of specifying the layout of an output line, but it was felt that the use of a visible character to denote blank characters between fields would be an advantage. This is especially so if the report layout is to be designed by the user whilst sitting at a terminal, rather than pre-planned using squared paper. It is far more
difficult to count the number of blank spaces between fields than it is to count the number of characters in a field.

Secondly, in the design of a problem specifying system for the generation of COBOL programs using the PG/1 macro processor, the numeric designation of control levels would be advantageous (3.7.4).

3.7.4 PG/1 Macro processor limitations

The main deficiencies in the PG/1 macro processor which were highlighted during the Filetab to COBOL generation study were as follows:

1. No facility for ordering alphabetic characters.
2. No subscripted variables.

In addition a number of desirable enhancements to the PG/1 macro processor facilities were identified, these included:

1. The provision of an overlay mechanism.
2. Provision of macro-time facilities for subfile input/output.
3. Provision of extraction facilities for long strings.

The PG/1 macro processor deficiencies became particularly apparent during the development of the macro-time statements for generating the COBOL paragraphs associated with control break processing (Appendix III Section 8). The Filetab alphabetic control characters could be compared only in an equal/not equal situation, and the lack of subscripting facilities prevented the use of table look-up as a means of ordering them. This resulted in the use of a large number of macro-time IF statements during the processing to determine the next control level in the Filetab job hierarchy.

The Filetab to COBOL translation study also served to highlight areas where enhancements to the macro processor
facilities would enable COBOL programs to be generated more effectively. The large number of macro-time statements required to process the various Filetab directive and parameter records strained the macro processor's internal areas for tables and stacks to the utmost (Appendix I Section 1). A significant increase in the core storage requirements would not ease the situation as, in many computer installations, the macro processor would be relegated from on-line mode to background batch mode. This points the need for some form of overlay facility.

A great weakness of the Filetab to COBOL system is the need to subdivide it into three phases, generating, filing and synthesising. This could be overcome if new macro-time statements to exploit the existence of the macro subfiles were designed and implemented. The ability to read, insert and delete specific subfile records from within a macro definition would prove a powerful tool. So too would be the implementation of macro-time statements to enable the output stack, containing the generated statements, to be emptied and the contents filed in a specific subfile.

The macro processor has facilities for the extraction of character strings from the input buffer, namely the macro-time statements CALLFSTR and CALLVSTR. These are, however, limited to eight characters, the maximum number which can be stored in a local or global variable. This meant that the text information from the parameter records of the *TITLE and *HEAD directives had to be extracted piecemeal (Appendix III Sections 5 and 6). This points the need for the design and implementation of improved macro processor facilities for handling character strings.
3.8 SUMMARY, APPRAISAL AND CONCLUSIONS

To an extent some of the difficulties outlined in the preceding sections are due to the structure and form of the macro processor used for this project and the nature of the COBOL language. However, the problem probably admits no really simple solution because of the complexity of the interlacing of the program order and the generating order of the COBOL statements, the very variable size of the items generated and the volume of the code to be generated. It would, of course, probably be possible to design a special purpose translator to carry out the translation efficiently for the particular language chosen for the project. The volume of code generated when producing COBOL programs necessitates the use of backing stores rather than main storage. This being so, the existing macro processor at least offers the advantage of a close integration between macro facilities, file handling facilities and system commands which control the operation of the system.

The work described in this chapter represents part of the feasibility study for this part of the project, since once the overall structure is defined it is relatively easy to fill in all the details (Appendix III).

The processor designed as the result of these investigations is a valuable outcome of the project in the sense that a system for 'growing programs' and synthesising general program structures is applicable to any target language.

The method used for the generation of COBOL programs using the PG/1 macro processor is in marked contrast with that used in the generation of FORTRAN programs [5]. In FORTRAN the volume of code generated for each routine was significantly less and, because it is a relatively unstructured language, the ordering of
generated statements did not present undue problems. Only very small sections of original FORTRAN code were generated in the main program, otherwise extensive use of pre-prepared subroutines was made. These subroutines were specially parameterised using the macro facilities so that they could be used to generate a wide variety of different subroutines for use in different situations.

The most important conclusion which may be drawn from the Filetab to COBOL generation study is that, from the experience gained and with some modifications to the macro processor, this approach can be used to create a system for translating a problem specified in a non-procedural language into a program specified in a procedural language. Secondary conclusions are that the generated COBOL is of reasonable quality and that a COBOL program is much more difficult to generate than a FORTRAN program (c.f. [5]).
CHAPTER 4

ENHANCEMENTS FOR THE PG/1 MACRO PROCESSOR

4.1 INTRODUCTION

The results reported in Section 3.7, as a result of experience with developing a Filetab to COBOL program generator, highlighted areas where enhancements to the PG/1 macro processor would enable COBOL programs to be generated more effectively. In order to specify a processor adequate to permit the completion of the planned project the following main areas were identified for improvement:

1. The introduction of an overlay facility to reduce the core storage demands of the macro processor, and thus make adequate space available for the macro processor's internal stacks and tables.

2. The provision of macro-time statements to permit the reading, writing, insertion and deletion of records in the macro processor's disc based subfiling system.

3. The implementation of macro-time statements which allow the PG/1 macro processor's output stack, containing generated statements, to be emptied when desired and the contents printed or filed in a specified subfile.

4. The extension of the macro processor facilities for character manipulation.

The desirability of improving the PG/1 macro processor facilities in the areas mentioned above was reinforced when the practical considerations for the design and implementation of the computer dialogue, proposed in Chapter 5, were taken into account.

The following paragraphs of this chapter specify in greater detail the facilities which would extend the scope of applications for the PG/1 macro processor. The subsequent implementation of
most of the proposals led to the development of the PG/2 macro processor. The formal specification of the enhancements proposed for the PG/2 macro processor are given in Appendix IV. (The implementation of the enhancements did not form part of the candidate's work on this project.) Section 4.8 gives an assessment of the PG/2 macro processor and the chapter concludes with a summary.

4.2 REDUCTION OF CORE STORAGE DEMANDS

The validation of input in the Filetab to COBOL generation study described in Chapter 3 was minimal. In contrast, extensive validation of the user's responses to the computer dominated dialogue described in Chapter 5 is required, before the information they contain can be used to generate COBOL statements. This is particularly true of the user's specifications for the selection of data, report layout design and own code processing. Extensive validation of responses is also required in the dialogue with the Data base Administrator, during the setting up and maintenance of the Catalogue of File descriptions (5.2 and 6.3). The macro-time statements for input validation and for the output of diagnostic messages when errors occur is very demanding in terms of storage requirements. Fortunately, the Catalogue maintenance and problem specifying dialogues of the COBOL generating system can each be divided into relatively independent stages. This is comparable with the way in which each Filetab directive was independently processed in the project described in Chapter 3. The development of a macro overlay facility was therefore a practical proposition from the COBOL generation point of view.

The request to the person maintaining the macro processor for the provision of an overlay facility led to the formulation of
the CHAIN feature (Appendix IV Section 1.6). When used in a macro
definition it causes, during the second pass of the macro
(Appendix I Section 2), the contents of one or more macro
definitions to be loaded into core and for one of the macros to be
called. The newly loaded macro definitions replace those
previously in core. A control parameter allows the user to
specify whether or not string variables and/or global variables
are to be initialised in addition to the reinitialisation of the
macro processor stacks.

The following annotated example illustrates the use of the
CHAIN feature:

```
%FILE,0,FILE2%
%DEF STAGE2

%END %DEF STAGE3
%END %DEF STAGE1

%FILE,0,FILE2%
%DEF STAGE2

%END %DEF STAGE3
%END %DEF STAGE1

[CHAIN0,FILE2]
[STAGE2]

Define and execute the STAGE1 macro.
```

During the second pass in the execution of STAGE1, the CHAIN
statement and the one following cause the contents of subfile
FILE2 to be loaded into core and macro STAGE2 to be called.
Macros STAGE2 and STAGE3 replace STAGE1 in core. The zero value
for the control parameter indicates that global and string
variables are to remain unchanged, but that the macro processor
stacks are to be reinitialised.

The CHAIN facility is extensively used during problem
specification in the COBOL generating system (7.1).
4.3 REDUCTION IN THE USE OF LABELS

As macro definitions become larger and more complicated the number of statement labels used becomes greater, but the label table of the macro processor has only a limited capacity. The larger macro definitions tend to make more use of the routine facility provided by the OBEY and RETURN macro-time statements. The OBEY statement specifies the label of the first statement in the routine to be obeyed and also the label of the statement to which the return should be made. Frequently the return statement is that immediately following the OBEY statement. When this situation occurs this pair of instructions is extravagant in the use of labels. Some economy in the use of the label table can, in such cases, be effected by the provision of two new macro-time statements for calling routines, viz CALL and EXIT (Appendix IV Sections 1.1 and 1.10).

The CALL statement requires that only the label of the first statement of the routine to be executed should be specified. The EXIT statement ensures that the called routine returns control to the statement following the CALL statement. The examples which follow illustrate the differences in the use of these pairs of statements and show the savings made in the use of statement labels.
The CALL and EXIT macro-time statements are used extensively to control the execution of routines in the macros of the COBOL generating system, e.g. routines PROMPT, SEARCH etc (Appendix VI).

4.4 ACCESS TO SUBFILES

The PG/1 macro processor only permits access to the disc based subfiling system by means of System commands, e.g. %LOAD, %FILE etc, which may not be used within a macro definition. This necessitated the use of three phases, viz Generation, Filing and Synthesis, in the Filetab to COBOL generation study (3.3.4).

In order to avoid a similar inelegant state of affairs in the self tutorial COBOL program generator, the provision of subfile handling macro-time statements was of prime importance. This requirement was emphasised by both the need to store and access the large volume of instructional text and the need to extract
information from the Catalogue of File descriptions. To meet these requirements five new macro-time statements were implemented to enable specific subfile records to be accessed and manipulated:

- **SELECT** (Appendix IV Section 1.19)
- **FREAD** (Appendix IV Section 1.11)
- **REPLACE** (Appendix IV Section 1.17)
- **INSERT** (Appendix IV Section 1.14)
- **DELETE** (Appendix IV Section 1.8)

The SELECT statement designates the subfile to which subsequent record handling macro-time statements refer. The named subfile remains selected until superseded by that given in the next SELECT statement executed.

In the other four macro-time statements the required record is referenced by its sequence number within the subfile. The FREAD, REPLACE and INSERT statements permit, respectively, the reading, replacement and insertion of a record. In the first instruction a designated string variable is used to receive the data from the subfile record, while in the two latter instructions the string variable is used as the source of the replaced or inserted data. The DELETE statement requires that only the number of the record to be deleted should be specified.
The following annotated macro outline illustrates these new macro-time statements in use:

```
%DEF FILECHANGE
%  SELECT DICTIONARY
%  DELETE 2
%  FREAD 8, #SO2
%  REPLACE 8, #SO3
%  INSERT #LV1, #SO1
```

- Select the DICTIONARY subfile for amendment.
- Delete the second record.
- Read the data from the 8th record into string variable #SO2.
- Replace the 8th record by the data in string variable #SO3.
- Insert a new record containing the data in string variable #SO1 before the record whose number is contained in local variable #LV1.

%END

The Catalogue maintenance macros (6.3) make extensive use of these new macro-time statements.

### 4.5 EMPTYING THE OUTPUT STACK

COBOL is a verbose language and even a request for a simple report requires the generation of a fair number of COBOL statements. Not only is the macro processor's output stack of limited capacity, but the generated COBOL statements are destined for different macro definition subfiles (7.3.4). In the PG/1 macro processor the system macro %SAVE is the only means of emptying the output stack and directing the disposal of its contents, but it cannot be used from within a macro definition. The interspersion of %SAVE commands between calls to macro definitions was rejected in the Filetab to COBOL study (3.3). With the provision of the CHAIN facility in the PG/2 macro
processor, which facilitates the automatic loading and calling of macro definitions, the use of %SAVE is not, in any case, possible until the end of the macro chain has been reached. In order to avoid both the overflow of the output stack and the need to have three phases, as in the Filetab to COBOL generation study, additional stack emptying facilities are required.

In order to satisfy the above requirements two new macro-time statements, PRINT and SAVE, were formulated (Appendix IV Sections 1.16 and 1.18). The proposed statements have a syntax and semantics which are consistent with other facilities currently provided by the PG/1 macro processor.

The PRINT statement causes the output stack to be emptied and the contents printed on the user's terminal if the macro processor is being used in on-line mode. If batch mode is used the stack contents is directed to the line printer.

The SAVE macro-time statement causes the output stack to be emptied and the contents directed to the specified subfile. A control parameter is used to indicate if the data is to be stored at the beginning of the subfile or appended after the existing records.
The following outline example illustrates the use of these new macro-time statements:

```plaintext
%DEF STAGE5  

 Generate statements in the output stack.

%SAVE,0,OUTREC  

 Empty the output stack and direct its contents to the beginning of the OUTREC subfile.

%SAVE,0,WRITEREC  

 Generate statements in the output stack.

%SAVE,1,OUTREC  

 Empty the output stack and append its contents to the OUTREC subfile.

%PRINT  

 Empty the output stack and print the contents.

%END
```

The implementation of these macro-time statements is discussed again in Section 4.8.
4.6 STRING HANDLING FACILITIES

4.6.1 String extraction

The proposals for the self tutorial COBOL generation system involve a considerable amount of character manipulation, both in the validation of user's responses and the extraction of relevant information for inclusion in the generated COBOL statements. The PG/1 macro processor's string extraction statements, CALLFSTR and CALLVSTR, permit the extraction of fixed and variable length character strings respectively. As the characters are extracted into local or global variables, eight is the maximum number of characters which may be extracted at one time. In the Filetab to COBOL generation study this restriction led to repetitive coding. For example, to extract the first 48 characters from the input buffer and use them in the generation of a COBOL statement to define the value of an alphanumeric literal, six uses of CALLFSTR are needed to place the 48 characters in six different variables.

To alleviate the above situation the PG/2 macro processor provides two new macro-time statements for extracting fixed and variable length strings, viz FSTR and VSTR (Appendix IV Sections 1.12 and 1.23). They are powerful instructions for not only do they permit the extraction of up to 72 characters into a string variable, but the source may be any string variable or the input buffer. The above mentioned problem may now be solved more concisely by using a string variable, say #S01, to store the extracted characters:

```
% FSTR1 ,1,48
```

The absence of the second parameter in the FSTR statement indicates that the characters are extracted from the input buffer. If the the required characters had to be extracted from string
variable #S02, then the above example would have been written as follows:

```
% FSTR1 #S02,1,48
```

The variable length string instruction, VSTR, is similar in form to FSTR except that the final parameter is the marker which, when encountered in the source data, terminates the operation.

### 4.6.2 Comparison operators

The PG/1 macro processor restricts to four the IF statement operators available for use with string operands, namely IN, AT, EQ and NE (Appendix I Section 5). In the PG/2 macro processor these are supplemented by four more operators (Appendix IV Section 1.13):

- LE (less than or equal to)
- LT (less than)
- GE (greater than or equal to)
- GT (greater than)

Together with the enhanced string extraction facilities, the new IF statement operators provide users of the PG/2 macro processor with powerful instructions for validating and ordering character string data.

### 4.6.3 Removing spaces from String valued variables

The PG/1 macro processor provides the SQUASH macro-time statement for the removal of spaces from string data in the input buffer. When the string data from which it is desired to remove spaces is in either a local or global variable six statements are required to achieve the desired effect:
The PG/2 macro processor does away with this inefficiency by means of a new macro-time statement, CALL SQUA (Appendix IV Section 1.4). This statement enables the above six statements to be replaced by the following statement:

\%
\% CALL SQUA \#LV1

### 4.6.4 Concatenating String valued variables

The concatenation of character data in a string variable is effected in the PG/1 macro processor by means of the DEPOSIT statement. For example, to concatenate the contents of local variable \#LV2 and string variable \#SO1 the following statement is used:

\%
\% DEPOSIT1 \#LV2\#SO2\$

If the concatenated data results in a short string of eight or less characters, it is uneconomic to use a string variable when a local or global variable would suffice. To cover this eventuality the PG/2 macro processor introduces the CALL DPOS macro-time statement (Appendix IV Section 1.2). This enables either text or the string contents of a local or global variable to be concatenated with the characters in another local or global variable. The result is always truncated if it exceeds eight characters, the maximum possible in such a variable. For example, to concatenate the contents of local variable \#LV1 with that of global variable \#GV2 we write:

\%
\% CALL DPOS \#GV2,\#LV1
If $\#GV2$ and $\#LV1$ initially contained 'ABCDEF' and 'GHIJKL' respectively, the resulting contents of $\#GV2$ would be 'ABCDEFGH'.

4.6.5 Interchanging characters

When manipulating a character string it is sometimes desirable to interchange characters. In order to meet this need the PG/2 macro processor provides two new macro-time statements, CALL SWAP and SWAP (Appendix IV Sections 1.5 and 1.21). The former is applicable to character strings in local and global variables, while the latter applies to the contents of string variables. Each statement requires three arguments:

1. The variable name or string variable number.
2. The character to be replaced.
3. The replacing character.

Single or multiple occurrences of the second argument character are replaced by a single occurrence of the third argument character. The SWAP statement can also be applied to the contents of either the input or output buffer. In the former case the first parameter is omitted, while in the latter case it is given the value zero.

If, given the string 'ABBCDBEF', it is desired to replace 'B' by 'G' so that the string becomes 'AGCDGEF', then one of the following macro-time statements can be used to achieve the desired result. The first statement is applicable if the given character string is in a local variable called $\#LV1$, while the second statement is used if the given string is in string variable $\#S01$.

```
% CALL SWAP $\#LV1$, B, G
% SWAP1, B, G
```
4.6.6 Extension of SQUASH and DEPOSIT

The SQUASH statement of the PG/1 macro processor enables spaces in the input buffer to be removed. In the PG/2 macro processor this facility is extended so that spaces can be removed from any designated string variable. The new form of the statement is:

```
% SQUASHn
```

where \( n \) denotes the number of the string variable from which spaces are to be removed (Appendix IV Section 1.20). If \( n \) is omitted then the operation applies to the input buffer as in the PG/1 version. If \( n \) is zero then the output buffer will be operated on. Global variable \(^{STC}\) is still used by the new macro-time statement to store the character count of the resultant string.

The PG/1 macro processor statement

```
% DEPOSITn <choice>
```

copies the chosen character string(s) into the designated string variable \(^{Sn}\), starting at the first available 'unfilled' character position. If \( n \) is zero the output buffer is used as the receiving string. The PG/2 macro processor extends this copying facility to include the input buffer as a receiving string (Appendix IV Section 1.9). This is achieved by omitting a value for \( n \) in the DEPOSIT statement.

4.6.7 Length of variables and strings

In the PG/1 macro processor the length of a variable or a string can only be determined by placing it in the input buffer, executing a SQUASH statement and interrogating the value in global variable \(^{GSTC}\). Besides requiring several operations, this
method is not useful if the string whose length is required
contains imbedded spaces, as they are not included in the character
count. To overcome these deficiencies the PG/2 macro processor
provides a new macro-time statement CALL LENV (Appendix IV Section
1.3). The name of the variable or string whose length it is
desired to know is supplied as a statement parameter, e.g.
\%
  CALL LENV #LV1
\%
  CALL LENV #S03
Global variable #GSTC is set to the length of the given variable
or string.

4.6.8 Copy arguments into strings

The PG/1 macro argument facility allows numeric arguments
to be assigned to either local or global variables, as shown in
the following example, where the / character is used to denote
the argument marker:
\%
  #DEF EXAMPLE1(/,/)  
  %   #LV1=/1:1/  
  %   #LV2=/1:2/  
  %   #LV3=/1:3/  
  
  %END
  %   EXAMPLE(5,4,8)
After the execution of the above macro call, local variables #LV1,
#LV2 and #LV3 will contain the values 5, 4 and 8 respectively.
It is not however possible to copy character string arguments into
string variables, because the DEPOSIT macro-time statement does
not include macro arguments among the possible choices. This
restriction is removed in the PG/2 macro processor (Appendix IV
Section 1.9), where it is possible to write statements of the type
shown in the following example:
After the execution of the above macro call, local variables /LV1 and /LV2 will contain the values 5 and 8 respectively. String variable /S01 will contain the character string 'PART-NUMBER'.

4.7 OTHER FACILITIES

During the preliminary investigations associated with the design of the self-tutorial COBOL generating system, three other desirable macro processor facilities were identified:

1. The provision of reserved global variables containing the current date and time and macro-time statements which enable their contents to be updated as required.

2. The extension of the arithmetic operation facilities to include multiply and divide operators.

3. To increase the number of subfiles in the PG/1 macro processor's disc based filing system, which is currently limited to a maximum of 30 subfiles.

4.7.1 Date and Time

As the Catalogue subfile (6.3) is updated 'in place', it is essential for security purposes to make 'back up' copies at regular intervals. Recovery procedures can, in the event of a failure, be carried out more readily if the date and time of creation of the subfile are known. To meet this need the PG/2 macro processor provides two new macro-time statements, DATE and TIME (Appendix IV Sections 1.7 and 1.22). These cause the current
date and time in character form to be stored in new global variables $\#GCDT$ and $\#GCTM$ respectively. When used in a macro definition, the DATE and TIME statements will cause $\#GCDT$ and $\#GCTM$ each to contain eight characters in the form DD/MM/YY (i.e. day, month, year) and HH/MM/SS (i.e. hours, minutes, seconds) respectively.

Each time the CATALOGUE subfile is updated, the contents of these global variables is included in the first record of the subfile. Their contents are also printed out for the Data base Administrator's information (Section 6.3).

4.7.2 Extended Arithmetic

The need for multiply and divide operators was highlighted by:

1. The calculations required to maintain the record counts in the CATALOGUE subfile.

2. The calculations to establish the current record number during the binary search of a subfile (Appendix VI Section 2).

3. The calculations associated with line counts in the preparations for showing the user a sample report page (7.25).

These are provided in the PG/2 macro processor, which uses the conventional computer language symbols * and / to denote the multiply and divide operators respectively (Appendix IV Section 5). The order in which the arithmetic operations in an expression are carried out is strictly from left to right. Thus after the evaluation of the following statement, $\#LV1$ will contain the value 7.

$\% \#LV1=2+8/3*4-5$
4.7.3 Main file facility

The COBOL report program generating system (Section 7.2) requires the use of a large number of subfiles for the storage of macro definitions, common data, dialogue text, etc. However, the design of the PG/1 macro processor allows a maximum of 30 subfiles in the subfile directory when its disc file is initialised. In order to overcome this limitation without having to completely restructure the subfiling system, the PG/2 macro processor has a facility which permits the use of any number of disc files, each containing a maximum of 30 subfiles.

The MAINFILE facility, implemented in the PG/2 macro processor in the form of a system macro (Appendix IV Section 3) and a macro-time statement (Appendix IV Section 1.15), allows the user to change the name of the currently referenced disc file at will.

Thus to change the name of the main file to FIRSTMASTER before loading and calling a macro definition, the user would use the system macro:

```
%MAINFILE,FIRSTMASTER
```

Within a macro definition, a user wishing to change the name of the currently selected mainfile to SECONDMASTER would use the following macro-time statement:

```
% MAINFILE,SECONDMASTER
```

4.8 ASSESSMENT OF THE PG/2 MACRO PROCESSOR

In the development of the macro processor from MP/1 through PG/1 to PG/2, the power of the processor has been increased. What was designed as a language preprocessor (to extend an existing procedural language) has been extended in an ad hoc way to:
1. Translate dialogue into FORTRAN.
2. Translate dialogue into COBOL.

The system, whose strengths lie in its string handling and expansion-time facilities, is clearly general purpose and reasonably flexible.

The weaknesses of the macro processor arise from its relatively low level language with ill defined syntax and poor diagnostics - the latter are a result of its history. As far as the present project is concerned the main weaknesses lie in the missing features identified below:

1. Subscripted variables.
2. Subroutines with names and parameters.
3. Separately compiled segments (CHAIN provides this in part).

The majority of the PG/2 macro processor enhancements described in Appendix IV were successfully and efficiently implemented, but the two following features require further consideration:

1. The facility to empty the output stack from within a macro definition.
2. The reading, replacement, insertion and deletion of subfile records.

4.8.1 Emptying the Output Stack

The proposals for emptying the macro processor output stack from within the body of a macro by two new macro-time statements, PRINT and SAVE, proved difficult to implement. The structure of the PG/1 macro processor is such that material is only transferred to the output stack when the %END statement which terminates the macro body is encountered. In order to implement the proposed
SAVE and PRINT macro-time statements, considerable restructuring of the macro processor was found to be necessary. The manpower and financial resources were not available to undertake this major reorganisation in the PG/2 version of the macro processor used for the practical work of this project.

4.8.2 Access to subfile records

Five macro-time statements, SELECT, FREAD, REPLACE, INSERT and DELETE, were implemented to provide facilities for accessing and amending the contents of subfiles within the PG/2 macro processor filing system.

Ideally these macro-time statements should be applied to subfiles with direct access or indexed sequential organisation. The subfiles within the PG/1 macro processor filing system are, however, sequentially organised. In order to avoid major changes to the subfiling system, the implementation of these macro-time statements in the PG/2 version of the macro processor was perforce inefficient.

To read the nth record of a subfile using the FREAD macro-time statement the following processing is required:

1. Open the subfile specified by the previous SELECT macro-time statement.
2. Read n records.
3. Return the nth record to the user.
4. Close the specified subfile.

The implementation of the REPLACE, INSERT and DELETE macro-time statements relies on the fact that a subfile can be read and written at the same time, providing that the operations do not refer to the same bucket. By provision of an internal buffer
large enough to hold the contents of 1 bucket (128 words) plus a little work space, this situation is avoided. Also the macro processor filing system has two 128-word buffers of its own, one for read and one for write. The macro-time statements for amending the contents of the nth subfile record requires the following processing:

1. Open the subfile specified by the previous SELECT macro-time statement.
2. Read and write all records up to the nth record.
3. Replace, delete or insert the nth record as required.
4. Read and write the rest of the subfile records until the end of the file is reached.
5. Close the specified subfile.

For subfiles containing 5 or 6 records this method of implementing access to records is acceptable. However, for larger subfiles, especially the Catalogue of File descriptions and those containing the instructional text, the delays are unacceptable. For example, the instructional text for the first stage of the problem specifying dialogue occupies 44 records in the DIAL1 subfile. The processing in the STAGE1 macro (7.8) consists mainly of reading the DIAL1 subfile records and outputting their contents on the user's terminal. During the execution of the STAGE1 macro the output of these 44 lines of text on a 10 characters per second teletype took approximately 11 minutes. The interval between the output of lines lengthened noticeably towards the end of the subfile. An output time of 5 minutes would have been more acceptable, as this would have kept the teletype almost continuously in operation.

It is appreciated that the terminal response times and access to disc files in a multiuser operating system are dependent
on the resource demands of other users, and that not all the printing delays can be attributed to the inefficient implementation of the macro-time statements. Nevertheless, the latter are significant enough to reduce the effectiveness of the COBOL report program generating system based on the existing PG/2 processor.

4.9 SUMMARY, APPRAISAL AND CONCLUSIONS

The macro processor enhancements specified for the PG/2 version (Appendix IV) provide only the basic facilities required for the generation of COBOL programs. The structure of the macro processor is such that, within the resources that were available, not all the enhancements could be provided and some could only be implemented inefficiently.

The lack of some desirable features and the inefficient implementation of others tended to prolong the time taken over the practical development of COBOL generating macros. The lack of the basic stack emptying facility precludes the complete development of the proposed system beyond the fourth stage (6.2.2).

The conclusion is that, although not an ideal tool for the generation of COBOL programs, the PG/2 macro processor serves to identify the characteristics of an appropriate tool (Appendix XI).
CHAPTER 5
THE USER INTERFACE

5.1 INTRODUCTION

In this chapter the interface between the casual computer user and the COBOL report program generating system is discussed. The material presented covers the following topics:

1. The need to present the casual user with a simple view of his data which does not depend on a knowledge of its physical organisation.

2. The adoption of the relational approach so that the user may view his data in the familiar tabular form.

3. Desirable features which should be incorporated into a dialogue designed for the casual user.

4. Aspects of the problem specifying dialogue which require more complicated user responses - conditions for data retrieval, report layout design and own code processing.

5. The gathering of users' comments on the system as an aid to improving the system.

6. The description of a model problem, the dialogue for which is available for viewing as part of the 'help' facility.

The chapter is supported by Appendix IX (which contains the dialogue for the model problem) and concludes with a summary and appraisal.

5.2 USER'S VIEW OF THE DATA

In order to generate a COBOL program the details of the physical and logical organisation of the data files to be used by it must be supplied to the program generating system. This section discusses how this can be done without burdening the casual user with the need to supply technical information and, at
the same time, allow him to take a 'simple' view of the data.

In common with report generators all COBOL programs require
details of the form and organisation of the external data on which
they are to execute. In the Filetab report generator the user
supplies these details in the 'FILE and 'INL directives (Appendix
II Sections 3.1 and 3.2). In a COBOL program similar information
has to be specified in the Environment and Data divisions. The
information required includes file medium, hardware assignment,
recording mode, file labels, block and/or record length, data
record names, record structure, descriptions of fields within
records, etc.

Despite the fact that many of the above COBOL data
specifications have default values, as do those for Filetab, it is
unreasonable to expect a non-programmer to understand the physical
organisation and storage of the data available for use, nor to be
able to supply even the essential details to any system which
generates COBOL programs. Not only will a non-programmer not
understand the physical organisation of a file, but he will have
difficulty in the concept of a record and with the organisation
of data into records and records into files. It is clear,
therefore, that some attempt to divorce the logical view of the
user's data from its physical organisation and storage must be
made, if a COBOL program generator, for use by a non-programmer,
is to be an effective and useful tool.

The separation of the physical organisation of data from its
logical organisation is one of the things which is achieved by a
Database system. A Database is defined by Martin [7] as:

'A collection of interrelated data stored together with
controlled redundancy to serve one or more applications; the data
are stored so that they are independent of programs which use the
data; a common and controlled approach is used in adding new data
and in modifying and retrieving existing data within the database. A system is said to contain a collection of data bases if they are disjoint in structure.

Different users have different views of the data within the same database. This is dictated partly by the security restrictions which may be applied to sensitive data. Thus each database user is provided by the Data base Management System with the following:

1. His requirements for access to data defined as a submodel, which is just part of the complete data model of the database.

2. A language with which to specify his requirements for processing the data.

The database user is concerned only with the logical organisation of that portion of the data available for his use, while the physical organisation of that data and of the complete database is the responsibility of a Data base Administrator. The responsibilities of a Data base Administrator range much wider than decisions on the storage structure and access strategy of the data in the database. Other responsibilities will include deciding on the information content of the database, maintaining a data dictionary (defining the data, its source, uses, ownership, etc.), liaising with users, defining security checks and validation procedures, ensuring that a strategy for back up and recovery exists in event of computer failure, monitoring the performance of the data base system and responding to requirement changes.

Data base systems are generally classified into three major categories:

1. Hierarchical.
2. Network.
3. Relational.
In a Hierachical data base a file contains records some of which are subordinate to others in a tree structure. The highest level record in the hierarchy is called a 'root'. All records except the 'root' are related to only one record in the tree structure and that is on a higher level than themselves, e.g.

```
Level 1
   Root

Level 2
```

```
Level 3
```

In a Network data base system a relationship between records exists such that a 'child' record can have more than one 'parent' record, e.g.

```
Level 1

Level 2
    Parent

Level 3
    Child
```

A Network structure can, however, be represented as a hierachical tree structure or structures but with redundant elements.

In contrast with the Hierachical and Network data base systems, a Relational data base gives the user a simpler view of his data. He sees it as a table and a set of relations between tables rather than as a linked record structure. This is particularly valuable as the tabular view of the data is familiar to even a non-programmer. The tabular view of a Relational data base is discussed further in Section 5.3.

Four main conclusions applicable to a COBOL generating system for use by non-programmers may be drawn from a preliminary
study of data base systems:

1. The user should not be aware of the physical organisation of the storage of the data files.

2. Access to sensitive data should be protected by some security mechanisms.

3. Only data relevant to the solution of the current problem should be described in the generated COBOL program.

4. The user's view of the data should be kept as simple as possible by adopting a relational view, and he should only be aware of the organisation of the data relevant to his application.

The above requires the setting up and maintenance of a catalogue of data file descriptions, to which the COBOL generating system can gain access by means of security passwords. Further discussion of this aspect follows in Chapter 6.

5.3 RELATIONAL APPROACH

This section shows how the 'flat' files used by a COBOL program lend themselves to a relational data base approach. Thus the user may take a tabular view of his data when describing his request for a report program to the COBOL generating system.

The relational approach is based on the mathematical theory of relations. Date [10] quotes the following definition of a Relation:

'Given sets \( D_1, D_2, \ldots, D_n \) (not necessarily distinct), the relation \( R \) on these \( n \) sets if it is a set of ordered \( n \)-tuples \( \langle d_1, d_2, \ldots, d_n \rangle \) such that \( d_1 \) belongs to \( D_1 \), \( d_2 \) belongs to \( D_2 \), \( \ldots, d_n \) belongs to \( D_n \). Sets \( D_1, D_2, \ldots, D_n \) are called the "domains" of \( R \). The value of \( n \) is called the "degree" of \( R \).'

The entanglements that build up in complex hierarchical or network structures can be avoided in the relational data base by
**Figure 5.1 STAFF Relation**

<table>
<thead>
<tr>
<th>DEPT-NO</th>
<th>EMPLOYEE-NO</th>
<th>EMPLOYEE-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>2056</td>
<td>BROWN C.W.</td>
</tr>
<tr>
<td>01</td>
<td>5021</td>
<td>JONES B.</td>
</tr>
<tr>
<td>01</td>
<td>8028</td>
<td>SMITH L.</td>
</tr>
<tr>
<td>02</td>
<td>1019</td>
<td>ADAMS M.</td>
</tr>
<tr>
<td>02</td>
<td>2022</td>
<td>BLACK J.</td>
</tr>
<tr>
<td>02</td>
<td>3027</td>
<td>FINLAY A.</td>
</tr>
<tr>
<td>02</td>
<td>4045</td>
<td>GREEN I.N.</td>
</tr>
<tr>
<td>02</td>
<td>6007</td>
<td>NEAL S.</td>
</tr>
<tr>
<td>03</td>
<td>7054</td>
<td>PORTER M.B.</td>
</tr>
<tr>
<td>03</td>
<td>8034</td>
<td>SMITH P.J.</td>
</tr>
</tbody>
</table>

**Figure 5.2 PERSONNEL Relation**

<table>
<thead>
<tr>
<th>EMPLOYEE-NO</th>
<th>SEX</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1019</td>
<td>F</td>
<td>20</td>
</tr>
<tr>
<td>2022</td>
<td>M</td>
<td>56</td>
</tr>
<tr>
<td>2056</td>
<td>M</td>
<td>30</td>
</tr>
<tr>
<td>3027</td>
<td>M</td>
<td>60</td>
</tr>
<tr>
<td>4045</td>
<td>M</td>
<td>38</td>
</tr>
<tr>
<td>5021</td>
<td>F</td>
<td>19</td>
</tr>
<tr>
<td>6007</td>
<td>M</td>
<td>23</td>
</tr>
<tr>
<td>7054</td>
<td>F</td>
<td>20</td>
</tr>
<tr>
<td>8028</td>
<td>M</td>
<td>42</td>
</tr>
<tr>
<td>8034</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.3 Join of STAFF and PERSONNEL Relations**

<table>
<thead>
<tr>
<th>DEPT-NO</th>
<th>EMPLOYEE-NO</th>
<th>EMPLOYEE-NAME</th>
<th>SEX</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>2056</td>
<td>BROWN C.W.</td>
<td>M</td>
<td>30</td>
</tr>
<tr>
<td>01</td>
<td>5021</td>
<td>JONES B.</td>
<td>F</td>
<td>19</td>
</tr>
<tr>
<td>01</td>
<td>8028</td>
<td>SMITH L.</td>
<td>F</td>
<td>42</td>
</tr>
<tr>
<td>02</td>
<td>1019</td>
<td>ADAMS M.</td>
<td>F</td>
<td>20</td>
</tr>
<tr>
<td>02</td>
<td>2022</td>
<td>BLACK J.</td>
<td>M</td>
<td>56</td>
</tr>
<tr>
<td>02</td>
<td>3027</td>
<td>FINLAY A.</td>
<td>M</td>
<td>60</td>
</tr>
<tr>
<td>02</td>
<td>4045</td>
<td>GREEN I.N.</td>
<td>M</td>
<td>38</td>
</tr>
<tr>
<td>02</td>
<td>6007</td>
<td>NEAL S.</td>
<td>M</td>
<td>23</td>
</tr>
<tr>
<td>03</td>
<td>7054</td>
<td>PORTER M.B.</td>
<td>F</td>
<td>20</td>
</tr>
<tr>
<td>03</td>
<td>8034</td>
<td>SMITH P.J.</td>
<td>M</td>
<td>29</td>
</tr>
</tbody>
</table>
the technique of 'normalisation', which was originally designed and advocated by Codd [11]. Although Codd's principles relate to the logical or user's view of the data and do not apply to its physical representation, it is fortuitous that many COBOL programs use physically 'flat' files which lend themselves to the relational approach. Thus the flat file may be regarded as a relation and the fields within the records of the file are regarded as domains.

The user of a Relational data base need not, however, be concerned with the three-level concept of normalisation as propounded by Codd. It is only necessary to present the user with the familiar two-dimensional concept of a table of data. Thus a relation is represented by a table in which the entries for a column are items from the same domain. The rows of the table, referred to as tuples, contain a set of data items relating to one entity, i.e. attributes. In order that each tuple can be uniquely identified it must have at least one key; that is, at least one attribute of a tuple must be designated as a key to ensure the tuple's unique identity.

Figure 5.1 illustrates the user's view of a relation called STAFF, which is defined over the domains DEPT-NO (department number), EMPLOYEE-NO (employee number) and EMPLOYEE-NAME (employee name). It requires two keys DEPT-NO and EMPLOYEE-NO to identify a unique tuple. Similarly, Figure 5.2 illustrates the user's view of a relation called PERSONNEL, which is defined over the domains EMPLOYEE-NO, SEX and AGE. The PERSONNEL relation has EMPLOYEE-NO as a key domain. As the STAFF and PERSONNEL relations have a common key domain, they can be joined to present the user with the view of the data shown in Figure 5.3.

A 'query language' is essentially the means by which the user specifies what he wants to do with the data, i.e. extract
information from the data base. For a relational data base the query language may be founded on either relational algebra or relational calculus. In the former, the user specifies the sequence of relational algebraic operations required to achieve the desired result. In the latter, the user states the result required in terms of relational calculus. The notations and relative merits of these two methods are discussed by Date [10], where he tends to come out in favour of the relational calculus approach, as exemplified by Codd's ALPHA language [12].

There are clear similarities between the report generator system and the relational approach to a data base, for in the former the user specifies the nature of the report he requires but not the sequence of operations required to achieve it.

5.4 DIALOGUE DESIGN

The way in which the user expresses his request for a report varies from one query language to another. In the MARK IV query language [13] the user fills in a standard request for information form which is key punched and processed to produce the report. Terminal based query languages make use of English words and vary in complexity depending on the user's skills and experience. For example, IQF (Interactive Query Facility) [14] is intended for relatively straight forward queries from a terminal by non-specialist users. In contrast GIS (General Information System) [15] is a high level programming language which permits more elaborate data searches and manipulations. It is intended for off-line queries by an experienced user. In this section desirable features of a dialogue with a casual user are considered in relation to the PG/2 macro processor COBOL generating system.

In an outline description of Rendezvous, a query formulation subsystem, Codd [16] draws attention to the fact that the casual
user cannot be expected to be knowledgeable about computer programs, logic or relations, nor can he be expected to learn an artificial language. Codd goes on to identify seven main steps which should be borne in mind during the design stage of a dialogue for use with a casual user.

Two steps outlined by Codd for his Rendezvous system are beyond the scope of the PG/2 macro processor being used to generate COBOL programs, because they make use of text analysers and synthesisers. Word transformers and access to large vocabularies are also required. These steps cover the following points:

1. The translation of the user's source statement into an internal high level language, e.g. data sublanguage ALPHA, in order to detect semantic incompleteness, incompatibility or ambiguity.

2. The restatement of the user's query by the system so that the user can check that the system has correctly interpreted his request.

The PG/2 macro processor could be used to provide Codd's third step but it would be inefficient:

3. The provision of a definition capacity for terms used during the formulation of the query.

A better approach would be to allow the Data base Administrator to define, at the user's request, some 'standard' macros which the user could then invoke in order to perform standard tasks.

The remaining four points made by Codd about dialogue design are, however, applicable to the COBOL generating system using the PG/2 macro processor:

4. Select a simple data model.

5. Introduce clarification dialogue of bounded scope.

7. Employ multiple choice interrogations as fall back.

The fourth point is concerned with the user's view of the data. The user's view should have enough structure to enable him to identify rapidly and concisely the part of the data base which interests him. Codd concludes that the relational view provides the right balance between too much and too little structure.

While the fifth point above specifically refers to a system with a very free format for query formulation, the principle of clarification can be applied to the more confined situation of a computer dominated/ user dialogue for use with the COBOL generating system. By asking follow up questions, the generating system can establish the user's requirements more precisely. Clarification can also be achieved by the insertion of examples in the explanatory text output by the COBOL generating system. The tutorial approach can be taken further by the provision of a 'help' facility when the former proves inadequate. The options available and the way in which the 'help' facility can be invoked are covered in detail in Section 6.2.3. They include a restart procedure and the facility to view the dialogue which solves a predefined model problem.

The sixth point above is particularly applicable to a COBOL program generating system. For while the computer/ user dialogue, which defines the problem to be solved, takes place in the foreground mode, the generation and execution of the COBOL program, which is effectively the data base search stage, are designed to operate in background batch mode (Figure 6.1).

Although Codd in his Rendezvous system, with its text analyser, synthesiser and dialogue control mechanisms, envisages the use of multiple choice interrogation as a fall-back measure, this approach is fundamental to systems with less capacity for analysing user's replies. From the casual user's point of view
the smaller the amount of typing required of him by the system, the fewer will be the keying mistakes he is likely to make. The approach adopted in the COBOL generating system is to reduce, as far as possible, the multiple choice to only two options, one of which is designated the default option.

The three following topics cannot readily be reduced to simple yes/no questions:

1. Specification of conditions for the retrieval of data.
2. Report layout design.
3. Own code processing.

Only the dialogue for these topics is examined further in the following three sections of this chapter. The remainder of the report requesting dialogue, which requires one word responses or poses simple yes/no questions, is illustrated by the model problem dialogue in Appendix IX.

The key features of a computer dominated dialogue may be summarised as follows:

1. The computer prompt - asking for a specific item of information.
2. The user's response - as brief as possible.
3. The follow up prompts - asking for more detail, i.e. clarification.
4. The error messages, help, back-up and recovery procedures.
5. The use of default options.
5.5 SPECIFICATION OF CONDITIONS FOR DATA RETRIEVAL

In order to solve a problem using the COBOL generating system the user will not always wish to view all the tuples retrieved from his relation(s). Some facility, which enables only the data from relevant tuples to be included in the report, must be provided. There are three main approaches by which a user can be encouraged to specify the conditions to be satisfied by data for inclusion in the printed report, viz:

1. Relational calculus.
2. Decision tables.
3. English language format conditional statements.

The use and ease of implementation of the approaches are discussed in the following paragraphs.

5.5.1 Relational calculus

For the casual computer user the relational calculus approach is highly inappropriate. Extensive tutorial dialogue by the system would be required in order to explain the uses of the unfamiliar boolean operators \( \land (\text{and}), \lor (\text{or}) \) and \( \neg (\text{not}) \). The use of the comparison operators \( =, \neq, <, \leq, >, \geq \) and the parentheses necessary to enforce the desired order of evaluation would also require much tuition.

Essentially, the non-programmer user simply cannot think in terms of relational calculus, nor can he begin to construct multi-term conditionals, let alone nested conditionals. The validation and translation into COBOL of such a relational calculus statement would also prove a long and relatively complicated task using the PG/2 macro processor facilities.
5.5.2 Decision tables

Decision tables provide a simple tabular representation of complex decision logic and were developed primarily as a device for man-to-man communication. Their structure is based on four quadrants, namely condition stub, condition entry, action stub and action entry (Figure 5.4). The simplest form of decision table is the limited entry table, where the rules are defined in terms of Y (yes) or N (no) entries against the condition stub entries. In the situation where a particular condition is irrelevant a dash (−) or an I is entered in the table. The actions to be taken if the rule is satisfied are denoted by X's against the action stub entries. For example, in Rule 2 of the decision table illustrated in Figure 5.4, action-1 is only taken if condition-1 is not satisfied and condition-2 is satisfied. Condition-3 is not relevant to Rule 2.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Condition Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>stub</td>
<td>Decision Rule 1</td>
</tr>
<tr>
<td>condition-1</td>
<td>Y</td>
</tr>
<tr>
<td>condition-2</td>
<td>-</td>
</tr>
<tr>
<td>condition-3</td>
<td>-</td>
</tr>
<tr>
<td>Action</td>
<td>Action Entries</td>
</tr>
<tr>
<td>action-1</td>
<td>X</td>
</tr>
<tr>
<td>action-2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.4 Limited entry decision table format

The techniques for the detection of redundancy, contradiction and incompleteness in decision tables are outlined by Pooch [17]. In this paper Pooch surveys the currently available decomposition and translation algorithms for communicating
decision table information from man-to-computer.

Decision tables are commonly used in report generators, e.g. Filetab [7], and in COBOL program generators, e.g. Cobra [18], but these software packages are tools for the programmer rather than the casual user.

Although decision tables give a reasonably natural description of selection logic, considerable effort would be required to develop a tutorial dialogue to instruct the casual user in the use of even limited entry tables. The problem of decision table validation would then have to be solved.

Two kinds of validation would need to be carried out on a decision table produced by an inexperienced user - format validation and content validation. The format validation would involve the checking of the condition and action stub entries and the column alignments of the condition and action entries. The content validation would involve checks for redundancy, contradiction and completeness.

The content validity checks and the algorithms for translating decision tables into COBOL program statements require the use of vectors and masking techniques, which are not available in the PG/2 macro processor facilities. This was the prime reason for the rejection of decision tables as a means of specifying conditional processing in the COBOL report generating system.

5.5.3 English language format conditional statements

To the casual user the adoption of English language format conditional statements is more natural than the alien notations of relational calculus or the less familiar decision tables. The use of easily remembered mnemonics for the condition operators enables a simple conditional statement, consisting of two operands separated by the operator, to be typed in on a single line by the
user, e.g.:

ACCOUNT-NUMBER .LT. TRANSACTION-NO (less than)
QTY-ON-HAND .NOT GT. 400 (not greater than)
PART-NAME .EQ. "NAILS" (equal)

In turn, this enables the string extraction facilities of the PG/2 macro processor to be exploited to advantage during the validation and translation into COBOL of each simple condition.

Although the use of parentheses would remove ambiguity and allow the creation of complex logical expressions, such expressions are both hard to form and hard to understand if they contain several (more than two) levels of nesting. The implementation and validation of such complex expressions would also be difficult using the PG/2 macro processor facilities. It was therefore not pursued. The non-provision of parentheses for conditional statements is characteristic of other COBOL generating systems, e.g. SURGE [19], which is not self-tutorial and uses fixed format card input.

Compound conditions can, however, be built up from simple conditions, without the use of parentheses, by the selection of appropriate AND/OR conjunctions. Essentially a system is devised where the computer is in control of the dialogue, and where the user is asked to provide only a small amount of information at a time. Simplicity is achieved at the cost of efficiency and flexibility, for after an OR conjunction the user has to be prompted to enter again previously typed conditions which still apply. e.g. The following complex conditional statement would have to be entered as eight simple conditions after selecting seven AND/OR conjunctions:

IF ((EYES .EQ. "BLUE") .OR. (EYES .EQ. "GREEN")) .AND.
   ((HAIR .EQ. "BROWN") .OR. (HAIR .EQ. "BLACK")) ... etc.
Thus:

IF EYES .EQ. "BLUE"
AND HAIR .EQ. "BROWN"
OR EYES .EQ. "BLUE"
AND HAIR .EQ. "BLACK"
OR EYES .EQ. "GREEN"
AND HAIR .EQ. "BROWN"
OR EYES .EQ. "GREEN"
AND HAIR .EQ. "BLACK"

The above approach is adopted in the dialogue for the specification of conditions for data retrieval in Stage 7 of the COBOL generating system (7.14). The two routines used, Specify condition (Appendix VII Section 15.1) and Validate condition (Appendix VII Section 15.2), are generally applicable to other situations where the processing may be conditional, e.g. own code processing.

In order to improve the legibility and clarify the structure of a user's compound conditional statement, it would be possible to restate it on his terminal in an indented form and without the intervening computer prompts. Further the labour of entering a complex conditional statement could, in some cases, be reduced by adopting the convention that default responses, entered after the selection of the OR conjunction, imply that previously specified simple conditions are repeated (and these could be listed on the terminal).

An alternative approach, whose implementation was not attempted, would be a facility for defining sets of values and permit conditional operations on these sets. Facilities to re-define a set in terms of existing sets and/or elements would also be required. Using this approach the previous example of a
compound conditional statement could be written:

SET EYE-COLOUR = ("BLUE", "GREEN")

SET HAIR-COLOUR = ("BROWN", "BLACK")

IF
EYES .IN. EYE-COLOUR
AND
HAIR .IN. HAIR-COLOUR ... etc.

The more complicated the bracketing used in a compound conditional statement, the more difficult it becomes to write it out in terms of simple conditions. Ideally the computer's prompt and reply system should give the user some idea of how his conditional statement is structured. The user really needs to be able to 'test his logic' on some typical data records (6.7).

5.6 REPORT LAYOUT DESIGN

Consideration is now given to the format in which the user is prompted to enter the specifications for the various types of line in his report and the field editing facilities provided.

Although the COBOL generating system described in Chapter 6 offers the user the default option of allowing the system to format the layout of the required data, according to a set of simple rules, many users will want to exercise their freedom to specify their own layouts.

The amount of preplanning required to design a report layout depends on the report producing system, its input device and mode of operation. It can range from a simple sketch to a detailed design on squared paper.

The manner in which the actual report line formats, including the required editing and insertion of punctuation and text, is described to a report producing system falls into two main types.

The first type necessitates the transcription of a detailed
preplanned pictorial layout for each type of report line into statements. These give details of the starting position, width, type (numeric, alphabetic, text, etc) and editing requirements, if any, for every data field in the print line. The starting position of any field is quoted either relative to the preceding field or relative to the beginning of the line. These format specifying statements may be in a relatively free format, as in COBOL, or in the strict format of the more concisely expressed RPG type languages.

The second type of report line format specification uses the pictorial approach as exemplified by Filetab, where each character in the line format specification represents one print position within the line. The Filetab approach is very suitable for an interactive report generating system as the user can design the report layout in detail while sitting at a terminal. This is especially true if the terminal is a visual display unit. The preplanning stage is minimal and the tedious transcription of a line format into detailed field specification statements is avoided altogether.

Filetab uses a Field Specifying Character to denote the field type and as a means of describing the print positions to be occupied by the field in a line of the printed report (Appendix II). Predefined character ranges are used to denote the field type, for example the characters M N O P Q and R are used to denote control fields. A slightly less restrictive approach is adopted for the present COBOL generating system. For each data item to be printed the user is invited to select a label character from a list of permitted characters without reference to item type. The latter is determined by the generating system from one of its subfiles, which contains details of the user's data base submodel. The character list contains most of the
alphabet, the exceptions being those characters which are used to denote editing requirements. From the available characters which are shown below, there is a reasonable opportunity to choose an appropriate mnemonic for a data item:

A E F G H I J K L N O Q T U V W X Y Z

The editing facilities offered to users of the COBOL generating system have been chosen to provide a useful selection from those available in Filetab and COBOL. The aim has been to provide the user with enough editing facilities to meet the requirements of most reports. The editing facilities could be extended to meet any reasonable requirement, but the penalty would be more complex processing and possibly a more complicated user interface. A balance between extremes has been sought in the realistic facilities described below.

The actual editing facilities available for use with any item are determined by the type of data the item contains. Thus alphanumeric data items are always left justified in the allocated print positions and two editing characters are available for use. These are the insertion characters B and O, where B is used to denote the insertion of a blank character and O the insertion of the 0 digit. These two insertion characters are also available for use in editing numeric items.

The decimal point of a numeric report item is assumed to be after the right-most digit of the field specification, unless otherwise indicated by the presence of the full-stop character. The decimal point in the actual data to be output is always aligned with that implied or specified in the output format. The leading zeros of a numeric item appearing in a printed report are suppressed, unless use is made of the currency symbols, £ and $, the cheque protection character *, or the + and - signs. Fuller details of the rules governing the use of these editing characters
together with the comma insertion character and the report credit and debit signs, CR and DB, are given in Section 6.5.9.

It tends to be difficult to judge how many blank characters have been typed between the various fields when keying in a report line specification at a terminal. To overcome this problem the COBOL generating system asks the user to key the visible character # to denote inter-field spaces. This is required only before and between fields and not for the blanks at the right-hand end of a line. A print line specification such as that shown below would be used for a report line which might look as that shown beneath it:

```
#####NNNN###III###III#######K.KK
```

324 FOOT PUMP £9.07

The tutorial dialogue for prompting the user to enter the various heading, detail and totals line specifications is illustrated in Appendix IX.

Two other facilities to help the casual user and to reinforce the tutorial dialogue are provided by the COBOL generating system. The first enables the user to experiment with the use of editing characters, while the second permits the viewing of the layout for a complete report page.

The experimental editing facility enables the user to design an edit format and have it validated. The system then invites the user to type a value which, if valid, is edited into the desired format and output for the user's inspection. The following is typical of the dialogue produced when using this facility:

**USING A VALID LABEL CHARACTER PLEASE TYPE THE DESIRED OUTPUT FORMAT -£AAA.AA**

**PLEASE ENTER THE VALUE OF THE NUMERIC ITEM TO BE EDITED INTO THE ABOVE FORMAT**

27.87

**AFTER EDITING YOUR NUMERIC ITEM WILL LOOK AS FOLLOWS:**

£27.87
The facility to view the format of a complete page enables the user to get an overall impression of the report layout and, if necessary, to change his mind before running the generated program with real data. The aim is to give the user a view of the spacing between the page heading and sequence break headings and also the alignment of the detail, subtotal and total lines. All the visible space characters and literal delimiters are replaced by blanks, while all the item edit pictures remain unchanged. Sufficient detail lines are generated to fill a page containing an example of each other type of report line. Thus the view presented to the user is analogous to the system analyst's squared paper report layout sheet. Further details of this facility are given in Section 6.5.11.

5.7 OWN CODE PROCESSING

The user's data base does not always contain the data in exactly the form he requires, e.g. details of the quantity in stock and the unit value of various items are available, but the user wishes to report on the total value of the stock. In such cases the provision of own code facilities, which enable the user to specify the calculations needed for the required information, are useful. Consideration is now given to the own code facilities offered by the PG/2 macro processor COBOL generating system.

In order to supplement the basic processing facilities of the COBOL generating system described in Chapter 6, the user may insert own code processing at various places within the generated program, e.g. after a data retrieval or before writing a report line. The dialogue for own code specification falls into several parts and is a combination of:

1. Direct questions with yes/no replies.
2. Default options.

3. More complicated tutorial dialogue for the specification of the own code data manipulations.

Examples of dialogue to illustrate the own code facilities are to be found in Appendix IX Section 18.

The introductory stage of the own code dialogue sets out the various points during the processing of the user's data at which own code processing may be carried out. The user is prompted to indicate if he wishes to avail himself of the facility. Once the user has signified interest the tutorial dialogue gives details of the own code operations together with examples. Initially the system makes provision for four arithmetic operations - addition, subtraction, multiplication and division. One data transfer operation is also provided.

The COBOL language allows two formats for specifying arithmetic processing. The first is the powerful COMPUTE verb, which permits complicated formulae to be expressed using a combination of parentheses, data-names and arithmetic operator symbols (+, -, * and /). For example:

```
COMPUTE AVERAGE = ( VALUE-1 + VALUE-2 ) / 2.
```

The second approach, although more verbose, allows the step by step evaluation of items by considering only one arithmetic operation at a time. It also has the advantage of clearly defining where intermediate values, if any, are to be stored, e.g.:

```
ADD VALUE-1, VALUE-2 GIVING SUM.
DIVIDE SUM BY 2 GIVING AVERAGE.
```

For ease of implementation and ease of use by the casual user, an approach similar to the second of the two COBOL forms was adopted for use in the COBOL generating system. The validation of complicated statements of the COMPUTE type would be difficult to implement using the PG/2 macro processor facilities. This is
because they may consist of arithmetic expressions nested by parentheses and require more than one line of keyed input when entered by the user. Problems similar to those outlined in Section 5.5 where complex conditional statements were considered would arise.

The main advantage of adopting the single operation per user statement approach is that the user can develop the calculation, step by step, while sitting at the terminal. There is no need to preplan the coding of complicated formulae, e.g.:

MULTIPLY ITEM-COUNT BY ITEM-COST GIVING NET-PRICE
MULTIPLY NET-PRICE BY VAT-RATE GIVING VAT
ADD NET-PRICE, VAT GIVING SALE-PRICE

The simplified forms of the COBOL verbs ADD, SUBTRACT, MULTIPLY, DIVIDE and MOVE can all be expressed in one line of output without undue restrictions on the number of characters allowed for domain or temporary item names. This enables the string extraction facilities of the PG/2 macro processor to be exploited during the validation of the user's own code statements. Once validated these user coded statements are used as the basis for the generation of COBOL statements for the report producing program.

In general the data names in the user's statements require further qualification to make them unique for use in the generated program. This is because, although the user takes the relational view in two dimensional tabular form, the actual items are extracted from various files and reorganised into tuple form.

When a user has taken advantage of both the own code and accumulated totals options, the COBOL generating system provides additional instructional text to explain how limited use of the totals may be made in the own code statements. This is achieved by instructing the user to preface data names by a special character to denote the current total for an item instead of the currently retrieved item value. In all situations where own code
processing is allowed, the user is invited to state if his own code is conditional and, if so, to specify the conditions.

The validation and generation of the COBOL statements from the user's own code entries is covered in more detail in Section 6.6.4.

5.8 USER FEEDBACK

For any particular problem the information to be gathered from the user by the COBOL generating system remains essentially the same, but a flexible approach to the dialogue by which the information is gathered should be maintained. One advantage of using a macro processor for the COBOL generating system is that it should be possible to revise the dialogue easily and/or provide new facilities.

Different users come from different backgrounds to solve a wide variety of problems. The feedback of their comments on the ease of use and clarity of the dialogue is an essential factor in the development and maintenance of any system which adopts the tutorial approach. As an encouragement to the user to make comments while they are fresh in his mind, he is invited to type in comments after he has specified his problem requirements. The comments are stored in one of the PG/2 macro processor subfiles, whence they may readily be retrieved by a system designer. The users' comments may then be taken into account at the periodic reviews of the facilities available and dialogue effectiveness, which should play an important part in the maintenance of the COBOL generating system.

The introductory dialogue for the use of the comments facility is illustrated in Appendix IX towards the end of Section 18.
Figure 5.5 Sketch of the required report layout

1. Report title
2. Page heading
3. Sequence break heading
4. Detail line
5. Sequence break subtotal
6. Totals output
5.9 DEFINITION OF A MODEL PROBLEM

The following model problem was designed to serve two main purposes. First, it is used in Appendix IX to illustrate the text of the computer dominated user/system dialogue produced by the COBOL generating system described in Chapter 6. Secondly, all users of the COBOL generating system are made aware of the model problem's existence prior to starting to use the system. Should they experience difficulty in defining the requirements of their own problem to the system at any stage, they can view the model problem dialogue at the corresponding stage, by means of the %HELP facility (6.2.3).

5.9.1 Model problem

At the end of the 1977 calendar year, the directors of the X.Y.Z. Manufacturing Co. Ltd. planned to give a bonus to all employees who would have completed 10 or more years of service with the company. The managing director therefore required a list by department of all employees who would be eligible for the bonus. He wanted the list to give details of employee number, employee name, date of joining the company and length of service in years and months. For each department and for the company as a whole, he also wanted to know the number of employees eligible and their average length of service. Employees joining the company during the month of January in any year were allowed to count that whole year as part of their service.

Figure 5.5 illustrates the layout of the required report. The following two paragraphs describe the data base details and the additional data items which have to be created and computed for inclusion in the report.
5.9.2 Data base details

The data needed to provide the required report is contained in three relations which would already exist, having been created for other purposes.

In order to gain access to the required information, the user must have been supplied with the passwords which permit access to the level 4 data in the DEPARTMENT and STAFF relations and the level 3 data in the PERSONNEL relation. (Level 1 is the highest security level used for the protection of sensitive or confidential data.)

The details of the three relations are as follows:

<table>
<thead>
<tr>
<th>DEPARTMENT relation</th>
<th>Domain names</th>
<th>Security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT-NO</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>DEPT-NAME</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>DEPT-LOCATION</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>DEPT-MANAGER</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Sequence key DEPT-NO

<table>
<thead>
<tr>
<th>STAFF relation</th>
<th>Domain names</th>
<th>Security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT-NO</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE-NO</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE-NAME</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Sequence keys DEPT-NO, EMPLOYEE-NO

<table>
<thead>
<tr>
<th>PERSONNEL relation</th>
<th>Domain names</th>
<th>Security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE-NO</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SEX</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>MARITAL-ST</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>AGE</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JOIN-YEAR</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>JOIN-MONTH</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>JOIN-DAY</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SALARY</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TAX-CODE</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Sequence key EMPLOYEE-NO
5.9.3 Additional data items

Some own code processing is required to solve the model problem since length of service and average service have to be computed from the date of joining information. In order to carry out these calculations and to count the number of eligible employees, the user has temporarily to extend the data base by creating the following numeric items:

- L-O-S-YEARS
- L-O-S-MONTHS
- NO-EMPLOYEES
- AV-SER-YEARS
- AV-SER-MONTHS
- WORK-ITEM
- MONTHS-SERVICE

The names chosen are quite arbitrary, the only conditions being that they must not duplicate existing names in the user's data base, not exceed 15 characters in length and satisfy the rules for COBOL data names. WORK-ITEM provides a useful means of storing intermediate results during the own code processing.

A compound domain JOIN-DATE is also created by concatenating the data items from the JOIN-DAY, JOIN-MONTH and JOIN-YEAR domains.

A listing of the dialogue for the solution of the model problem is given in Appendix IX. The listing is separated into separate stages to make cross referencing easier. The dialogue has been manually fabricated for those problem specifying macros whose development is incomplete or have not been implemented (7.1). No attempt has been made to illustrate the many error messages which invalid user responses might provoke.
5.10 SUMMARY AND APPRAISAL

In a viewpoint on computer language design, Tucker [20] describes a 'very high level' language as one which enables the user to express, in familiar terms, what is to be done rather than how it is to be done. The dialogue in which the casual user specifies his request for a report to the COBOL generating system is akin to a 'very high level' language, in that the user specifies the content and form of his report, but does not specify how the report is to be produced.

The casual user is presented with a simple view of his data which does not require knowledge of its physical organisation. The relational approach used in the dialogue allows the user to view his data in a familiar tabular form.

The computer dominated dialogue aims to instruct the casual user and prompt him to supply specific items of information as briefly as possible. This includes the use of default options and, in some instances, follow up prompts requesting more detail. Some aspects of the dialogue do not lend themselves to brief responses, e.g. data selection, report layout design and own code processing. The dialogue for these aspects of the report specification is influenced by the constraints imposed by the PG/2 macro processor facilities and the type of terminal available for system development. The dialogue also provides facilities for help, trial editing, resuming the dialogue at an earlier stage, model problem and sample page viewing and gathering users' comments.

In a discussion of data base interrogation languages, Martin [7] concludes that it is desirable that they should operate on standard data structures by means of an on-line computer initiated dialogue. Other desirable characteristics listed by Martin are that the dialogue responses should be in a fixed format using two
dimensions and entered by means of a light pen. The COBOL report program generating system uses standard flat data files and, subject to the limitations of the PG/2 macro processor and a teletype terminal, the dialogue goes some way towards exhibiting the desirable characteristics advocated by Martin.

Although the dialogue illustrated in Appendix IX shows that it is feasible to specify a request for a report by means of a computer dominated dialogue, a complete operational system is necessary before the dialogue can be fully appraised. Only when the users' reactions to the clarity, naturalness and simplicity of the dialogue have been analysed can its effectiveness be assessed. Ways in which assessment data can be gathered are considered in Section 8.5.
CHAPTER 6

SYSTEM SPECIFICATION

6.1 INTRODUCTION

The COBOL report program generating system, designed for implementation using the PG/2 macro processor, adopts a relational view of the data and uses a computer dominated dialogue for communicating with the casual user. The design and system specification details are presented in this chapter under five main headings:

1. The system overview which sets out the main benefits of the COBOL report program generator and relates its overall structure to that of the host computer's operating system.

2. The design and maintenance of the Catalogue containing descriptions of the data files.

3. The processing of system input in the form of the Catalogue and the user's data files.

4. The report output facilities including the form of the output file and the design of a report page.

5. The data manipulation facilities which include the provision of additional data items, both system and user created, the totalling facility and the use of own code and conditional statements.

The chapter concludes with a summary and appraisal.

6.2 SYSTEM OVERVIEW

This section sets out the main aims and benefits of the COBOL report program generating system and relates its overall structure to the operating system of the host computer. Features which contribute to more efficient computer usage are noted. The stages into which the problem specifying dialogue is segmented
are identified and the provisions of the 'help' facility are outlined.

6.2.1 Aims, benefits and proposals

The main aims which have influenced the overall design of the COBOL generating system are as follows:

1. Use of man/computer computer dominated dialogue as the basis for generating COBOL programs to solve a limited range of data processing problems.

2. Make it 'easier' for the casual user to use the computer by adopting a tutorial approach and taking a relational view of the data.

3. Make a contribution towards helping the user to use the computer more efficiently by reducing wastage in terms of time and resources at the man/computer interface.

4. To offer some of the benefits of a data base management system.

The range of data processing problems which may be solved using the COBOL generating system has initially been restricted to the generation of reports using sequentially organised flat data files. The system has, however, been designed with a view to determining the potential for enhancement (and avoiding unnecessary restrictions) so that it may then be used to solve a wider range of problems. There is potential for solving problems which may require the creation and/or updation of data files whose organisation can include direct access.

The benefits of the data base management system approach accrue in that:

1. The physical organisation of the data files does not concern the user as he has only to take a relational view of the data.
PG/2 Environment: running under time-sharing

Foreground jobs

Operating system, utility programs, COBOL system running under batch

Background jobs
2. Sensitive data is afforded some protection by a system of passwords. Different users will have access to different data items depending on the security level of the passwords they are allocated.

The system to generate a report falls into four main parts:

1. The creation and maintenance of the catalogue of data file descriptions.
2. The problem specifying dialogue.
3. The generation of the COBOL program.
4. The compilation and execution of the generated program to produce the required report.

Only the second part is carried out by the casual user, although he may initiate the execution of part three. Figure 6.1 shows how the above tasks relate to each other, to the PG/2 macro processor and to the operating system of the host computer.

As aids to helping the user make more efficient use of the computer the following features are incorporated:

1. Only the problem specifying dialogue is carried out in foreground mode. Once the problem details have been 'filed' they are passed across to the system so that the generation, compilation and execution of the COBOL program may be carried out as a batch job in background mode.

2. The user is allowed some opportunity to experiment with the design of report page layouts and field editing. He is also able to view a stylized page before committing himself to a full production run.

3. The dialogue is check pointed to allow it to be restarted at a previous stage.

4. There is a 'help' facility which offers the user various options including the chance to view the dialogue for solving the model problem and the opportunity to change his mind about
earlier decisions by restarting the dialogue at a previous check point.

6.2.2 Check point stages

The problem specifying dialogue is segmented into eighteen check pointed stages which, in association with the %HELP facility, enable the user to change his mind about decisions made in a previous stage. Each stage covers a new aspect of the problem specification and is introduced by some instructional text, the first line of which introduces the stage. The user's problem may not require the use of all the stages which are listed in order below:

1. Introduction
2. Password dialogue
3. Data base submodel
4. Selection of relevant domains
5. Temporary extensions to the data base
6. Data base inconsistencies
7. Selection of data for retrieval
8. Report layout introduction
9. Editing
10. Detail line(s) specification
11. Extra data items - date, time, page number
12. Report title specification
13. Page heading specification
14. Sequence break heading specification
15. Subtotal line(s) specification
16. Total line(s) specification
17. View of sample report page
18. Own code processing
In general, the stages are carried out in serial order although, in Stage 17, there are provisions for repeating any of the line format specification stages if the user is dissatisfied with his design.

6.2.3 Asking for help

The 'help' facility may be invoked by typing the five character string '%HELP' in reply to any prompt at any stage during the problem specifying dialogue.

Initially the following options will be available to the user when he enters a plea for help:

1. Abandon the dialogue.
2. Restart the dialogue at the beginning of the current stage or at a previous stage.
3. To view the dialogue for the corresponding stage of the sample problem and resume processing at the beginning of the current stage.
4. To experiment with editing data and then resume the dialogue at the beginning of the current stage. (This option is only available to users who have already reached Stage 9 where editing is introduced.)

As each stage depends on information gathered in an earlier stage it is, with the exception of the report layout specification stages, not possible to provide facilities for repeating just one stage.
6.3 THE CATALOGUE SYSTEM

This section describes the security mechanisms which protect the catalogue of file descriptions and identifies the basic processes which are required to maintain it. The catalogue structure is described together with the constraints which influenced the design of record formats. The macro definitions used for catalogue maintenance are described in outline. The chapter concludes with a brief appraisal of the catalogue handling system.

6.3.1 Catalogue maintenance and security mechanisms

The catalogue of file descriptions is located in the PG/2 macro processor CATALOGUE subfile and is maintained by a number of macro definitions which carry out a dialogue with the user. As it is envisaged that the catalogue will be maintained by an experienced user, the Data base Administrator, the dialogue does not attempt to be tutorial. Extensive validation of the user's responses is, however, carried out.

The catalogue is protected by a single password and the data items within each file description are protected by a series of passwords. The latter define different levels of access and are dual purpose for they limit access to data and permit creation of different views of the data.

Up to twelve levels of access are permitted for file descriptions, but this is an arbitrary limit which can readily be extended should the need arise. Any user may have read access to a file description providing he is able to supply a valid password for that file. The amount of information to which access is allowed is determined by the security level of the password supplied. A password for one security level
automatically grants access to that level and all levels below it.
The highest security level is 1, so the password for level 3
grants access to data with security levels 3 to 12 inclusive. If
different users are allocated different passwords for the same
file, they will have different views of its contents.
Initially macro definitions are provided for:
1. Creating an empty catalogue.
2. Changing the catalogue password.
3. Inserting or deleting file descriptions.
These can readily be supplemented to provide facilities for
printing out file descriptions, changing file passwords, etc.
The provision of such facilities, however, adds nothing to the
research project because no original ideas or techniques are
involved.
Only users able to supply the catalogue password may insert
or delete file descriptions. In the case of a deletion the
highest security level password must also be known. Changes to
a file description are effected by a deletion followed by an
insertion. The method of record modification is a simple,
minimum effort approach. It would be easy to define a macro to
access, edit and replace a record, but time consuming to produce
it and unnecessary for this project.

6.3.2 Structure of the catalogue

The catalogue of file descriptions is located in the
CATALOGUE subfile of the PG/2 disc based filing system and
contains three classes of record:
2. File name records.
3. Field description records.
The two latter classes contain more than one record type and are
Figure 6.2 Catalogue structure

Catalogue control record
File name record type 1
File name record type 10
File name record type 2
File name record type 3
Field description record type 1
Field description record type 2
Field description record type 1
Field description record type 2

Field description record type 1
Field description record type 2
File name record type 1
File name record type 10
File name record type 2
File name record type 3
Field description record type 1
Field description record type 2
e tc.

First file description

Second file description
repeated for each file described in the catalogue. As many of the
dfields in the catalogue records contain variable length data a
commas is used as the field separator. Figure 6.2 illustrates
the catalogue structure.

Catalogue control record:

The CATALOGUE subfile contains only one record of this type
and it is the first record in the subfile. The control record
contains the following information:

Count of file descriptions in the catalogue
Count of all records in the catalogue
Catalogue password
Number of security levels used for protecting access to
field descriptions
Date and time when the catalogue was last updated

The format details of the control record are given in Appendix V
(Section 1.1).

File name records:

Each file described in the catalogue has at least three file
name records. If the catalogue security system extends beyond
four levels then a fourth record will be present. File name
records are always followed by the field description records for
the same file. The file name records contain the following
information.

Type 1

- File name
- Count of the number of fields described in the
catalogue
- Passwords, one for each security level up to the
fourth
Type 1C record

{ Passwords for security levels 5 to 12, if used,
otherwise this record is omitted }

Type 2 record

Description of file contents

File medium
Maximum block size
Maximum record size

Type 3 record

File label, if applicable
Access mode
Organisation, if applicable
Length of symbolic key, if applicable
Number of fields forming the symbolic key or sort key

Although the COBOL generating macros currently restrict processing
to sequential files, the file name records have been designed in
anticipation that this restriction will be eased at a later date.
Format details for the file name records are given in Appendix V
(Sections 1.2 to 1.5).

Field description records:

For each field in a file record which is likely to be used
by the system a pair of field description records is set up.
Not all fields in the record need be made available to the system,
but access to those which are is protected by the set of multi-
level passwords. Fields may overlap or redefine one another.
The inclusion of the password and redefine facilities means that
different users will have different record descriptions in the
generated COBOL program.

Currently fields are restricted to DISPLAY usage. In ICL
1900 COBOL parlance this means that each character or digit in a
field occupies one ICL six-bit character. Although the COBOL language allows three categories of data (alphabetic, alphanumeric and numeric), it was felt that for the purposes of this study the two latter categories would be adequate. These initial restrictions do not preclude the inclusion of other usages or classes of data at a later date.

The field description records contain the following information:

- **Field name**
- **Security level**
- **Start position**
- **Length of field**
- **Type 1 record**
  - **Field type**
  - **Decimal point location, if applicable**
  - **Key field indicator**
  - **Sequence order for key fields**
  - **Domain indicator**

- **Type 2 record**
  - **Description of field contents**

The domain indicator serves to show whether or not the contents of this field throughout the file can be considered to form a domain. For practical purposes, this means that the contents of this field in every record of the file is unique.

Format details for field description records are given in Appendix V (Sections 1.6 and 1.7).
6.3.3 Constraints on the design of catalogue records

The formats for each type of record in the CATALOGUE subfile are given in Appendix V. The restrictions on field size or contents were imposed for two main reasons:

1. Those imposed by the ICL 1900 series COBOL compiler or operating system, e.g.
   - 30 characters for a file name
   - 12 characters for a file label
   - 120 characters for the maximum field size
   - 9 sort keys
   - 64 characters for the length of a symbolic key

2. Those which seem reasonable to cover most practical applications, e.g.
   - File, record and field counts
   - Number of security levels

The contents of certain fields is temporarily restricted by the facilities implemented in the initial versions of the macros. For example, binary fields are excluded and only numeric and alphanumeric fields are catered for. The catalogue records are designed to permit the addition of enhancements without changes to the record formats.

The notable exception to the above is the limitation of field names to 14 characters when COBOL would permit the use of up to 30 characters. The computer dominated dialogue with the user restricts responses so that they will fit on one line of the teletype or visual display unit. The problems of user error and response validation are much greater if continuation lines are allowed. The user's condition specifying and own code statements are those where the length of the field name (domain name) becomes critical.

Of the 72 positions normally available on a terminal device, one is taken up by the operating system's invitation to type...
character. A further character is sacrificed in order to force the invitation to type on to the line following a request for information (Appendix VI Section 1 - PROMPT routine). A maximum of 70 characters is then available for the user's response.

Although the own code statements, which are those where space for domain (field) names becomes most critical, would allow up to 16 characters, this has been reduced to 14 to cope with the following situations:

1. When a non-key domain occurs in more than one relation (file) it is necessary to distinguish between occurrences as their contents may not be identical. Additional numeric characters are suffixed to each duplicate non-key domain name in the user's data base submodel to pad it out to 15 characters. For example, if the non-key domain DELIVERY-COST occurs in two relations, when the data base submodel details are displayed to the user it will be called DELIVERY-COST in the first relation and DELIVERY-COST22 in the second relation.

2. When a user opts to have totals accumulated for the contents of numeric domains (6.6.3) he is allowed access to the totals in own code statements (6.6.5). In order to distinguish between the total of items from a given domain and the a single item from the same domain, the domain name is prefaced by a * character when the total is referenced. For example, if a user has opted to have the contents of the domains DELIVERY-COST22 and MILAGE accumulated, he can calculate AVE-COST-MILE using the following own code statement:

DIVIDE *DELIVERY-COST22 BY *MILAGE GIVING AVE-COST-MILE

The restriction to 14 characters was not felt to be too inconvenient for the data base administrator when he devised meaningful mnemonics for field names. This is especially so as the catalogue includes a description record for each field.
6.3.4 Creation of an empty catalogue

The CREATE macro creates the control record for an empty catalogue (Appendix V Section 1). The macro carries out a short dialogue with the database administrator to gather details about the catalogue password and the number of security levels by which the data files to be described therein will be protected. The implementation details for the CREATE macro are described in Section 7.4.

6.3.5 Changing the catalogue password

The PASSCHANGE macro enables the database administrator to change the password in the catalogue control record. The macro carries out a brief dialogue during which the old password must be correctly entered before the new password may be specified. The implementation of the PASSCHANGE macro is described in Section 7.5.

6.3.6 Inserting and/or deleting catalogue file descriptions

A set of chained macros, the Librarian macros, is used to carry out the insertion and/or deletion of file descriptions in the CATALOGUE subfile. The macros are chained because of the large number of macro-time statements needed to gather and validate all the information needed to add a file description to the catalogue. In order to modify a file description it is first necessary to delete the old description before the new description is inserted (6.3.1).

Because the security processing for the insertion and deletion of a file description have common features in the checking of passwords and file names, it is convenient to carry out the catalogue amending processes by means of four macros -
LIBRARIAN, LIBPRELIM, LIBADD and LIBDELETE. Figure 6.3 illustrates the function and linkage of the macros. The implementation details for the Librarian macros are described in Section 7.6.

Figure 6.3 Librarian macros for catalogue maintenance

Password dialogue for catalogue access.

Dialogue to establish the user's option - add, delete or finish. When add or delete is selected the file name and password are requested before chaining to the appropriate processing macro.

Dialogue to gather information about the new file. Insert the description in the catalogue and update the catalogue control record.

Delete the file description from the catalogue and update the catalogue control record.
6.3.7 Appraisal

The catalogue system with its dual purpose passwords enables the COBOL report program generating system to present a relational view of the data base to the casual user. Enhancements to the COBOL generating system, which extend the range of acceptable field or file types, can readily be accommodated within the present catalogue format.

The macro definitions specified are the minimum necessary to maintain the catalogue for use by the COBOL generating system.

The Librarian macros for inserting and deleting file descriptions frequently access the CATALOGUE subfile and their run time efficiency is adversely affected by the inefficient implementation of the PG/2 macro-time statements for subfile access (4.8.2).

This problem aside, there is considerable potential for extending the range of facilities available to both the data base administrator and the casual user for accessing catalogue information.

6.4 INPUT PROCESSING

The description of the report program generator part of the system starts with this section which deals with input processing.

The use made of the catalogue of file descriptions and the implications of using more than one input file to solve the user's problem are discussed. The input to the report generator part of the system is summarised diagramatically in Figure 6.4.

6.4.1 Input file descriptions

Descriptions of all the files that may be used by the system are held in the CATALOGUE subfile which is maintained by the data base administrator (6.3). The COBOL generating system
Figure 6.4 COBOL report program generating system input summary

Data files on various media containing problem data (maximum number of files is 9)
carries on a dialogue with the user about relation names (file names) and passwords to determine which files are to be used by the generated COBOL program to produce the required report. From the information gathered during this dialogue a shorter version of the catalogue is written in the SUBMODEL subfile (Appendix V Section 2 and Figure 6.4). SUBMODEL contains details of only those files which contain problem data. Only fields to which the user has password access and which feature in the problem are described in SUBMODEL.

From the information in the SUBMODEL subfile all the COBOL file description statements can be generated without the user having to know anything about the physical or logical organisation of the data. The system designed can only generate COBOL statements for sequential access files on punched cards, paper tape, magnetic tape or disc. The files must be in ascending sequence and each record must have a unique key. The catalogue is, however, designed to hold descriptions of direct access files and those with keys in descending sequence. Processing to generate COBOL statements for such files can therefore be incorporated in the system design at a later date. The various restrictions on the system design are made necessary by the need to design a complete typical system (all features) and therefore no one feature can be followed in every aspect.

The two main usage forms common to all COBOL dialects are DISPLAY and COMPUTATIONAL, although there are frequently other usages which are machine dependent. In the initial system design all fields described in the catalogue are assumed to have DISPLAY usage. Only numeric fields can be used in arithmetic operations, but sequence key fields may be either numeric or alphanumeric.
6.4.2 Use of more than one input file

The data files to be used by the generated COBOL program were not specifically designed, created or maintained for use in a relational data base. They are therefore not necessarily in a suitably normalised form. In order to generate COBOL statements to control the reading of records and the matching of sequence keys when the report data is located on more than one file, some restrictions must be placed on the files which are acceptable. This problem is approached in three ways.

First, all non-key field names must be unique; only a key field name may be used in more than one file. This restriction is not imposed during catalogue maintenance (6.3), but is delayed until Stage 3 of the problem specifying dialogue (7.10), where it is automatically resolved. Duplicate non-key field names are made unique by appending characters to the name before it is displayed to the user (6.3.3). A user may notice that the names of domains in a relation change slightly from problem to problem.

In the second approach to the handling of non-normalised data files, the user may be made aware of restrictions, if the files he selects do not conform to certain key matching criteria. The system designed is capable of generating COBOL statements to match records in files which have the same sequence key fields, or whose sequence keys are an ordered subset of those in the file with the most keys, e.g.:

<table>
<thead>
<tr>
<th>File number</th>
<th>Record format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>KEY-A KEY-B KEY-C KEY-D non-key data fields</td>
</tr>
<tr>
<td>2</td>
<td>KEY-A KEY-B KEY-C KEY-D non-key data fields</td>
</tr>
<tr>
<td>3</td>
<td>KEY-B KEY-C non-key data fields</td>
</tr>
<tr>
<td>4</td>
<td>KEY-A non-key data fields</td>
</tr>
<tr>
<td>5</td>
<td>KEY-D non-key data fields</td>
</tr>
<tr>
<td>6</td>
<td>KEY-D KEY-E non-key data fields</td>
</tr>
<tr>
<td>7</td>
<td>KEY-E non-key data fields</td>
</tr>
</tbody>
</table>
Records from any combination of files 2 to 5 may be matched with records from file 1. Records from files 6 and 7 cannot be matched with records from file 1 as it does not contain KEY-E.

If the keys in the files selected by the user do not satisfy the file matching criteria, he is advised, during Stage 3 of the problem specifying dialogue, that his relations cannot be joined. The run is terminated unless the user wishes to respecify the relations to be used.

The third way in which the use of more than one data file can affect the generation of the COBOL program is in the handling of unmatched key values. As the data files are created and maintained by other programs, their integrity cannot be guaranteed. In Stage 6 (7,3) the user is therefore invited to select one of three ways of handling unmatched keys:

1. The unmatched records are ignored.

2. A dummy matching record is created in which all the numeric non-key fields are zero and all the alphanumeric non-key fields are blank.

3. The execution of the COBOL program is terminated.

In all cases the user is given the option of having an error message displayed by the COBOL program. The error message will give the file name and key values together with a brief comment. The user may, if he wishes, specify the text of the comment which, by default, consists of the word 'MIS-MATCH'.

Up to nine input data files can be handled by the COBOL generating system and each file may have up to nine key fields. Each record is assumed to have a unique key value.
6.5 REPORT OUTPUT FACILITIES

In this section the overall design of the report itself is discussed. The generated COBOL program assigns the report output to a line printer file, where each non-blank line of the report occupies one record within the file. The form and content of the report file records are largely under the user's control as, during the problem specifying dialogue, he is offered a comprehensive range of facilities to enable him to design the layout of a printed page.

Details of the report output facilities offered to the user and the means by which they are provided are described in the paragraphs which follow. These facilities consist of choosing the report page dimensions and the selection and design of various categories of output which may be used to enhance the presentation of the report data, e.g. title, page heading etc. The specification of print line formats and the facilities for editing the output data are also described. The section concludes with a description of the facility to view a styled sample page and the options available for redesigning it.

6.5.1 Page dimensions and categories of output

Although it is hoped ultimately to offer a choice between a narrow (72 characters) and a wide (120 characters) page the present implementation offers only a narrow page.

The format of the report lines and the number of lines per page are specified during a dialogue with the system. A page may contain between 40 and 99 lines with 60 lines being the default option. The report file records are specified under six categories:

1. Report title
Figure 6.5 Record category and field type summary

<table>
<thead>
<tr>
<th>Field type</th>
<th>Input File</th>
<th>Alphanumeric text literals</th>
<th>Pagename</th>
<th>Temporary User</th>
<th>Subtotal</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record category</td>
<td>N-T</td>
<td>T</td>
<td>Date</td>
<td>N-T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Report title</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Page heading</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Sequence break heading</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Detail line(s)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Subtotals</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Totals</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
2. Page heading
3. Sequence break headings
4. Detail lines
5. Subtotals
6. Totals

The user is prompted to design the format of records in each of the categories applicable to his problem. Sequence break headings are offered only if two or more sequence keys are used in the input file(s). Subtotals and totals are offered only if the automatic totalling facility has been requested (6.6.3).

Each category may consist of more than one line of output and include blank lines. The user may also specify the number of blank lines which precede the first line of an output category. The format of each output line is described by the user in a similar way to that used in Filetab (2.3). Text in the form of an alphanumeric literal may also be included in any report line. Further details of the allocation of field specification characters are given in Section 6.5.8.

Figure 6.5 summarises the types of field which may appear in each category of output. Descriptions of the four right-most field types are given in Section 6.6.

Although the wide page option would make full use of the line printer facilities, the narrow page has a number of points to commend it, especially as the user's terminal may be restricted to 72 characters per line:

1. When the generated COBOL program is executed with real data the output may, by means of job control commands, be directed to a filestore file. Later it may be recovered and listed on the user's terminal without the wrap round/truncation problems which occur when an overlength line is output on a terminal.

2. The narrow page computer produced report can readily be
bound with manually produced material on A4 sized paper.

3. It is proposed to offer the user the opportunity of seeing a dummy report page after he has completed the design. The wide page would have to be displayed on the user's terminal in two parts, the left-hand side of the page and then the right-hand side. Checking the page layout where the two parts meet would then present problems for the user.

A record description for each type of non-blank report line is generated in the WORKING-STORAGE section of the program in the form of a group field. The elementary fields of the group will depend on the contents of the print line specified. The WRITE ... FROM .... AFTER ADVANCING .... form is used in the Procedure division of the program to move the group contents to the report file record area and to control the line spacing when the report file is printed. The generated program uses Z-REPORT-FILE as the report file name and Z-OUTREC as its record area.

6.5.2 Report title

A title is optional, but if present it occupies the first page of the report. It consists of one or more lines some of which may be blank. The non-blank lines may include text, the page number, the date and the time. The page number of the title page is zero.

The user may specify on which line of the page the first line of the title is to appear. This must be a non-blank line and may not be more than half way down the page. By default the title will start at the line one quarter of the way down the page.

In the generated program the non-blank title line data descriptions are group fields called Z-TITLE-n, where n is the
number of the title line excluding intermediate blank lines, e.g. Z-TITLE-1, Z-TITLE-2 etc.

6.5.3 Page heading

A page heading is optional but if present it consists of one or more lines some of which may be blank. Any non-blank lines may include text, the date, the time, the page number, any temporary data item or any item from the currently retrieved set of data (i.e. the current tuple). Page numbering always starts at 1, even if the report title has been omitted. If a printed page heading is not required COBOL statements are generated which ensure that a skip to a new page (i.e. channel-1) is made as required without visible printing.

The user may specify on which line of the page the first line of the page heading is to appear. This must be a non-blank line and may not start more than 10 lines from the top of the page. By default the heading will start at line 1.

A page heading, either visible or invisible, is followed by the sequence break headings, if specified, for the current major to penultimate minor keys.

In the generated COBOL program the group data descriptions for the page heading lines are called Z-PAGE-n, where n is the number of the page heading line excluding intermediate blank lines, e.g. Z-PAGE-1, Z-PAGE-2 etc.

6.5.4 Sequence break headings

The user is offered the sequence break heading facility if his data files have two or more sequence key fields. A sequence break heading may be specified for each sequence key from major to penultimate minor and consist of one or more lines, some of which
may be blank. Text, temporary items or any item from the currently retrieved set of data may be included in any non-blank line.

The user may specify how many blank lines should appear between the previous line of output and the first line of the sequence break heading. The number of blank lines specified must be in the range 0 to 3, with 0 as the default value. The first line of the sequence break heading must be non-blank.

All the sequence break headings are printed at the top of each report page immediately after the page heading, if present. They are printed in major to penultimate minor key order. They may also appear lower down a report page when a sequence break in an item retrieved from a key field is detected. A sequence break at one key level will cause that key sequence break heading and also those at lower levels, if any, to be output. If, for example, 4 sequence keys are used in the input file, where level 1 is the major key, a sequence break at the level 2 key will cause the headings for key levels 2 and 3 to be output. Near the bottom of a report page sequence break headings are only output if there is also room to output at least one set of detail lines, otherwise a complete new page is taken.

In the generated program the group data descriptions for the sequence break heading lines are called Z-HEADm-n, where m is the sequence key level and n is the number of the non-blank line within that heading, e.g.:

Z-HEAD1-1, Z-HEAD1-2, ... major key heading
Z-HEAD2-1, Z-HEAD2-2, ... submajor key heading
6.5.5 Detail line(s)

For each retrieval from the input files, excluding those rejected by the selection conditions, one or more lines of output may be written on the report. A detail line may include text or any input field or temporary item. If more than one detail line is to be output for each retrieval, blank lines may be included.

The user may specify how many blank lines should appear between the previous line of output and the first line of detail, which must be non-blank. The number of blank lines specified must be in the range 0 to 3 with 0 as the default value.

The generated program data descriptions for detail lines are called Z-DETAIL-n, where n is the non-blank line number within the group of detail lines, e.g. Z-DETAIL-1, Z-DETAIL-2 for two lines of output per retrieval.

6.5.6 Subtotals

Subtotals output is offered only if the user has invoked the totalling facility and the input data files have two or more sequence key fields.

Subtotals are maintained for all totalling fields at all key levels except the minor key. If requested, subtotals are output when there is a sequence break in any non-minor key field. The subtotals are output in key level order from penultimate minor to major key. A sequence break in a key at an intermediate level will cause the subtotals for that key and all keys minor to it to be output. If, for example, 4 sequence keys are used in the input files, where level 1 is the major key, a sequence break in the level 2 key will cause the subtotal output at levels 3 and 2 to be printed. The level 3 output will precede that for level 2.

At each sequence key level, except the minor key, subtotal
output consists of one or more lines including blank lines. The first line must be non-blank and any non-blank line may contain text, non-totalling items, either temporary or from the last set of retrieved data, and the accumulated subtotal of any totalling item at that sequence key level.

At each subtotal level, the user may specify how many blank lines should appear between the previous line of output and the first line of the subtotal. The number of lines must be in the range 0 to 3 with a default value of zero.

Near the bottom of a report page a subtotal for any level is only output if there is room for all the lines of that level, otherwise it will appear on the next page of the report.

In the generated program the group data descriptions for the subtotal lines are called Z-SUBTOTALm-n, where m is the sequence key level and n is the number of the non-blank line within that subtotal, e.g.:

Z-SUBTOTAL1-1, Z-SUBTOTAL1-2, ... major key subtotal lines
Z-SUBTOTAL2-1, Z-SUBTOTAL2-2, ... submajor key subtotal lines

6.5.7 Totals

Totals output is offered only if the user has invoked the totalling facility. The totals output is written at the end of the report after the last detail record and subtotals records, if present.

Totals output consists of one or more lines and may include blank lines, but the first line must be non-blank. Each non-blank line may include text, non-totalling items, either temporary or from the last set of retrieved data, and the accumulated totals of any totalling items.

The user may specify how many blank lines should appear between the previous line of output and the first totals line.
This number must be in the range 0 to 3 and the default value is zero. If, near the bottom of a page, there is not enough room for all the total lines a new page will be used.

In the generated program the non-blank totals line data descriptions are group fields called Z-TOTAL-n, where n is the number of the total line excluding intermediate blank lines, e.g. Z-TOTAL-1, Z-TOTAL-2 etc.

6.5.8 Field specification

In order to design the layout of any category of report line the user is invited to select, from a given list, one alphabetic character for each field to be included in the report. A glossary mapping data field names to alphabetic label characters is thus created. When specifying the format of a report line the print positions to be occupied by a field are indicated by typing its label character in the desired positions (c.f. Filetab). Blank characters before and between fields are indicated by typing ' ’ characters. Blank characters at the right-hand end of a line needed not be typed.

Up to nineteen fields may be included in a report and their label characters may be selected from the following:

A E F G H I J K L N O Q T U V W X Y Z

The remaining characters of the alphabet are reserved for use by the editing facility (6.5.9) or to indicate where the date, time and page number fields are to print (6.6.1). Unlike Filetab, special types of field (e.g. key, transfer, totalling) are not denoted by special groups of characters. Instead the user is free to select a label character which may also serve as a mnemonic for a field name, e.g. N for NAME, A for ADDRESS.

The double quotation mark character (" ) is used to delimit
the text literal. It was chosen in order to avoid confusion with the single quotation mark character ('') which is used as a literal delimiter by the macro processor.

If more print positions are allocated to an alphanumeric field than the field actually contains the data is left justified in the specified positions and unused positions are filled with blanks. If too few print positions are allocated to an alphanumeric field then the characters at the right-hand end are truncated.

Unless the editing feature (6.5.9) is in use, the decimal point in the output format of a numeric item is assumed to be after the right-most digit. Leading zeros are replaced by blanks up to but not including the zero before the implied decimal point. The decimal point location in the field to be output is aligned with that implied in the output format. If more print positions are allocated than the field requires, the unused positions are filled with blanks. If too few print positions are allocated the high order digits will be truncated.

The names of the data fields are displayed on the user's terminal one at a time. As each field name is displayed the user indicates that it appears in the report by typing the character he has chosen as its label. The default reply, made by pressing only the 'Accept' key, indicates that the currently displayed field does not appear in the report.

The date and time from the additional data items (6.6.1) each occupy 8 characters and the user must indicate their print positions by typing the character strings DD/MM/YY and HH/MM/SS respectively. The page number, also an additional data item, occupies four characters and the string PPPP is used in the required print positions. Leading zeros in the page count will be suppressed when the value is printed.
In order to distinguish text from field label characters in a line format specification the text is enclosed in quotation marks. The quotation marks are replaced by blank characters when the line is printed.

Although not implemented, it would be appropriate and desirable to provide the user with the facility which enables him to inspect his glossary of labels. In response to one or more label characters he could be given the name of the data assigned to that character. Details of the label character, field name and field type (alphanumeric or numeric) are, however, written to the LABELTABLE subfile (7.15). This subfile is used during the validation of the line format specifications for the various categories of report record.

The model problem dialogue for Stage 8 (Appendix IX Section 8) contains examples of the assignment of label characters.

6.5.9 Editing

The editing facilities have been chosen to provide a balanced selection of those available in COBOL. Because the field editing requirements are entered during the specification of print line formats, the edit characters are present in the field label character strings which denote the print positions of the field. Each field label character string has to be translated into a COBOL data description with the appropriate PICTURE. The editing facilities for any field depend on its type, i.e. alphanumeric or numeric.

In order to help the casual user understand the editing facility it is proposed to allow some experimental editing before the line formats are entered or the generated program is run with real data (6.5.10). By means of the 'help' facility the user may also 'break out' of designing the report to do some more editing
experiments (6.2.3).

Alphanumeric field editing:

Two insertion characters are available for use with alphanumeric fields, they are the characters B and 0.

Insertion characters may be inserted anywhere in the string of label characters defining the print positions of an alphanumeric field. The 0 character will print as a zero in the output line, but the B character will be replaced by a blank character.

Numeric field editing:

The following characters are used to specify editing requirements of numeric fields:

£ % * + - . CR DB B O

The function of each of the above symbols and the permitted combinations are set out below.

£ and % are currency symbols which, if used, must appear once only at the beginning of the field format specification. The selected currency symbol will print in front of the most significant digit of the edited result. A currency symbol may be preceded only by a + or - character.

* is the cheque protection symbol. If used, this character must appear at the beginning of a field format specification. It will cause the unused high order print positions of the edited result to be filled with * characters. Only the + or - characters may precede it.

+ and - signs. A + or - sign may appear as the first character of a field format specification. If the sign of the data being edited is the same as the sign character then the sign character will print. If the data has the opposite sign to that in the edit format then a + sign will be replaced by a - and
a - will be replaced by a blank in the edited result. The sign character, if printed, will appear to the left of the most significant digit of the item providing that a currency or cheque protect symbol has not been used. If the currency or cheque protect symbols have been specified the sign will remain in the first position of the field.

CR and DB are report signs. Either CR or DB may appear in the two right-most positions of the field format. If the data being edited is negative then the report sign characters will print in the specified positions. If the data is positive then the two print positions occupied by the report signs will be set to blanks.

BO , and • are insertion characters which, with the exception of the B character, will print in the designated positions of the edited field. The B character will be replaced by a blank. The decimal point may not appear in the right-most print position of an edited field and must be preceded by at least one label character. The implied decimal point in the data is aligned with the decimal point in the print edit format. If insufficient print positions have been allocated on either side of the decimal point truncation will occur at the both ends of the edited result. If more print positions to the right of the decimal point have been allocated than are required, they will be filled with zeros. Insertion characters to the left of the high order digit are replaced by blanks.

6.5.10 Experimental editing

Although the text explaining the field editing facilities to the user contains examples, the editing rules are difficult to grasp at the first attempt. To alleviate this difficulty it is proposed to offer the user the opportunity to experiment with the
editing facilities before he designs his report format. The user is invited to type a field output format, either alphanumeric or numeric, using a valid label character and edit symbols. This is validated and, if found acceptable, the user is invited to enter a value to be edited. The edited result is then displayed on the user's terminal (5.6). This process may be repeated until the user is satisfied that he can design the field editing formats which will give the output he requires.

Figure 6.6 illustrates the use of the editing symbols and shows their relation to the COBOL edit PICTURE which would appear in the generated program.

<table>
<thead>
<tr>
<th>Label character</th>
<th>Field type</th>
<th>Edit format</th>
<th>COBOL PICTURE</th>
<th>Value of data to be edited</th>
<th>Edited result</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>A</td>
<td>KBKKKKOKK</td>
<td>XBXXXXOXX</td>
<td>C NUT4DE</td>
<td>C NUT4ODE</td>
</tr>
<tr>
<td>A</td>
<td>N</td>
<td>-£AAA,000</td>
<td>-£££9,000</td>
<td>-23</td>
<td>-£23,000</td>
</tr>
<tr>
<td>E</td>
<td>N</td>
<td>*EEEEOCR</td>
<td>****9CR</td>
<td>4795</td>
<td>**4795</td>
</tr>
<tr>
<td>F</td>
<td>N</td>
<td>FF,FFFF.FF</td>
<td>ZZ,ZZ9.99</td>
<td>1347.63</td>
<td>1,347.63</td>
</tr>
<tr>
<td>G</td>
<td>N</td>
<td>GGBGBGGG</td>
<td>ZZBZZBZ9</td>
<td>27641</td>
<td>27641</td>
</tr>
</tbody>
</table>
6.5.11 Sample page viewing

Because the formats for the various categories of output are entered during the course of a dialogue it is not easy to get an overall impression of the report page. Especially as the formats for the various categories of output are not entered in the same order that they would appear in the report, e.g. detail lines are specified before the page heading.

It is therefore proposed to offer the user the chance to see a sample of the report page layout before the generated program executes on real data. The user may then respecify the format for any category of output which is not to his liking and view again the sample report.

The output sample will consist of a title page, if present, and a report page containing one example of each of the output categories requested by the user except the detail category. This latter category will be repeated several times to give the impression of a full page. The output categories appear in their natural order, i.e. page heading, sequence break headings, details, subtotals and totals. The page layout display will enable the user to check the vertical alignment of heading, detail, subtotal and total data.

The lines in each category will be spaced according to the user's requirements and consist of the line format specifications as entered by the user with two modifications. All visible space characters (#) and all text literal delimiters (") will be replaced by blanks.

An example of the page layout display is included in the model problem dialogue for Stage 17 (Appendix IX Section 17).
6.6 DATA MANIPULATION FACILITIES

In this section the additional processing features provided by the system for the manipulation of data are described and their means of provision are outlined.

These features include the provision of additional data items by the system, the creation of temporary items by the user, the automatic totalling of user selected numeric data items and the provision of own code statements for manipulating data items. The facilities for extending the own code statements to access totals and subtotals are described. The section concludes by describing the condition specification facilities which permit the selection of report data and/or the conditional execution of own code processing.

6.6.1 Additional data items

In addition to the data items from the user's data base domains three others are available for inclusion in his printed report. They are the date and time at which the generated COBOL program begins to execute and the count of pages in the report. The user is introduced to these items in Stage 11 of the system (7.19) and their field specification strings are described in paragraph 6.5.8.

The COBOL program obtains the date and time from the operating system executive by means of ACCEPT statements in the initialisation section of the Procedure division. The ACCEPT statements refer to data items in the SPECIAL-_NAMES paragraph of the CONFIGURATION section. In the WORKING-STORAGE section the date and time are stored in items ZDATE and ZTIME which are part of the Z-TUPLE group. The generated COBOL program therefore contains the following statements:
CONFIGURATION SECTION.

SPECIAL-NAMES.
  *DATE IS Z-DATE *TIME IS Z-TIME.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 Z-TUPLE.
   02 ZDATE PICTURE X(8).
   02 ZTIME PICTURE X(8).

PROCEDURE DIVISION.
Z-INITIAL SECTION.
Z-PARA-1.

  ACCEPT ZDATE IN Z-TUPLE FROM Z-DATE.
  ACCEPT ZTIME IN Z-TUPLE FROM Z-TIME.

The page count is stored in the Z-PAGE-COUNT data item which is also part of the Z-TUPLE group. This variable is updated during the execution of the statements generated to control the taking of a new page (Appendix VIII Section 23).

6.6.2 Temporary data items

The system includes provision for the user to create temporary data items. These exist only in the WORKING-STORAGE section of the generated COBOL program, but may be included in the printed report. Their contents may be manipulated using the own code facility (6.6.4) and they fall into three types - numeric, alphanumeric and group. This facility is offered in Stage 5 of the system (7.12) and the model problem dialogue (Appendix IX Section 17) contains an example of its use.

Numeric items:

The system explains to the user how temporary numeric items may be needed to store results and intermediate results arising
from the own code facility, e.g. a quotient or a remainder. In
dialogue with the user the system asks for a description of each
temporary numeric item by prompting him to enter details of the
item name, size, decimal point location and, if desired, the
initial value. By default the initial value is set to zero. In
the generated COBOL program the item is assigned COMPUTATIONAL
usage.

The automatic totalling facility (6.6.3) is also available
for temporary numeric items.

Alphanumeric items:

The user is invited to create temporary alphanumeric items
which may be used with the own code facility. The system prompts
the user to describe the alphanumeric item by asking for details
of its name, size and, if desired, the initial value. By default
the initial value is set to be blank when the item is described in
the WORKING-STORAGE section of the COBOL program.

Group items:

Group items may be formed by concatenating data from two or
more other items, for example, three data items containing
respectively details of street, town and postcode may be
concatenated into a single group item called, say, ADDRESS:

```
  02 ADDRESS.
    03 STREET PICTURE X(20).
    03 TOWN PICTURE X(20).
    03 POSTCODE PICTURE X(10).
```

The items to be concatenated may be either numeric or alphanum-
numeric and come from any domain or temporary item. In the
dialogue they are referred to as compound domains. The system
carries out a dialogue with the user and prompts him to enter the
names of the items to be concatenated. Group items are assumed to
be alphanumeric and their size is the sum of the sizes of the concatenated fields. When the user has finished specifying the names of fields to be concatenated, the system prints out a message to tell the user the size of the items in the compound domain, i.e. the group item.

In the generated COBOL program, the MOVE statements which set up the group are executed after the file records have been matched and before any own code processing.

6.6.3 Totalling facility

The system provides for the automatic totalling of data in a numeric field of an input file or in a temporary numeric item should the user so desire. The accumulated subtotals and totals (6.5.6 and 6.5.7) are available for inclusion in the output report and may be used in own code processing (6.6.4).

If totalling is requested by the user, the system generates the data description statements in the WORKING-STORAGE section ready for the totals to be stored. Control totals are generated for each requested item at each sequence key control level except the minor key. A control total is also generated for the final totals of the run. When, for example, the input file(s) have three sequence keys and automatic totalling is required for numeric items COST and QUANTITY, the following statements would be generated:

```
01 Z-CONTROL-TOTALS.
  02 Z-LEVEL-2.
    03 COST PICTURE ...  Submajor key
    03 QUANTITY PICTURE ...
  02 Z-LEVEL-1.
    03 COST PICTURE ...  Major key
    03 QUANTITY PICTURE ...
  02 Z-LEVEL-0.
    03 COST PICTURE ...  Final
    03 QUANTITY PICTURE ...
```
All the items are given an initial value of zero and are described as COMPUTATIONAL. The PICTURE description will depend on the location of the decimal point in the field, but will always be signed and have the maximum size of 18 digits.

The use of the same data names at each control level enables the system to make use of the ADD CORRESPONDING statement in the Procedure division for the accumulation of totals, e.g.:

Z-PARA-ADD.
ADD CORRESPONDING Z-TUPLE TO Z-LEVEL-2.

The ADD statement is executed after any own code processing associated with the data retrieval and prior to the setting up of a detail line for the report.

The ADD CORRESPONDING statement is also used to add the totals of one level to the totals of the next level when a sequence change in a key field is detected. The SUBTRACT CORRESPONDING statement is then used to reinitialise the control totals at that level, e.g.:

ADD CORRESPONDING Z-LEVEL-2 TO Z-LEVEL-1.
SUBTRACT CORRESPONDING Z-LEVEL-2 FROM Z-LEVEL 2.

The totalling facility is offered to the user in Stages 4 and 5 of the system (7.11 and 7.12). The model problem dialogue for these stages contains examples of this facility (Appendix IX Sections 4 and 5).

6.6.4 Own code processing

Own code processing is optional and supplements the basic data movement and totalling features provided automatically in the generated program.

Own code processing allows the user to specify his own calculations and/or data movements for insertion at the following points in the generated program:

1. After the files have been opened but before any data is
2. Immediately after a data retrieval, i.e. after a set of matched records from the input data files have been read.

3. Immediately after a sequence break in any specified key, except the minor key, is detected.

4. Prior to printing any category of output lines present in the program, i.e. title, page heading, sequence break heading, detail, subtotal or total lines.

Five statements are provided whereby the user can express his own processing requirements. Four statements are for arithmetic purposes and one is for data transfer. Each statement permits one operation and is expressed in English or COBOL type language. By means of instructional text, with examples, the system describes the use of the own code statements. In addition, if the user has availed himself of the totalling facility, he is told how to access the subtotals at any key level or the final totals.

Any number of own code statements may be entered for inclusion at the permitted places in the generated COBOL program. The user is prompted to enter one own code statement at a time. After validation each statement is translated into a syntactically correct COBOL statement by the insertion of data-name qualifiers and punctuation. Before each own code statement the user may specify what conditions, if any, must be satisfied before it is obeyed (6.6.6).

Arithmetic statements:

There is one statement for each of the arithmetic operators (addition, subtraction, multiplication and division), and each contains an operator and three operands. The third operand specifies where the result of operating on the first two operands
is to be stored. Each statement is entered by the user as a one line response and takes one of the following forms:

ADD operand-1, operand-2 GIVING operand-3
SUBTRACT operand-1 FROM operand-2 GIVING operand-3
MULTIPLY operand-1 BY operand-2 GIVING operand-3
DIVIDE operand-1 BY operand-2 GIVING operand-3

When the DIVIDE statement is used, the user is asked if he wishes to save the resulting remainder and, if so, to specify the name of the item where it is to be stored. This must be of numeric type and may be a domain or temporary item.

Operand-3 may be the name of any temporary item or the name of a domain to which the item being calculated belongs.

Operand-1 and operand-2 may be either a numeric literal, the name of a temporary item or the name of a domain to which the item being calculated belongs, e.g.:

MULTIPLY QUANTITY BY PRICE GIVING COST
DIVIDE PENCE BY 100 GIVING POUNDS

Data transfer statement:

One data transfer statement is available for own code use and it takes the following form:

MOVE operand-1 TO operand-2

Operand-2 is the receiving item and may be the name of a temporary item or the name of a domain to which the item receiving the data belongs.

Operand-1 is the source item and may be a literal, a temporary item or the name of the domain to which the item being moved belongs. Operand-1 must be of the same type, i.e. numeric or alphanumeric, as operand-2.

When alphanumeric data is moved, the characters from the source item are transferred to the corresponding positions of the
receiving item, starting with the left-most. If the receiving item is the longer item then the excess characters in the receiving item are filled with blanks.

When numeric data is moved, the decimal points in the source item and receiving item are aligned. If there are fewer characters before or after the implied decimal point in the receiving item, then the excess characters in the receiving item are set to zero.

The above rules for data movement are in line with those for the COBOL MOVE verb with the added restriction that both operands must be of the same type.

The following are examples of own code MOVE statements:

```
MOVE 1.65 TO UNIT-PRICE
MOVE QTY-ON-ORDER TO QTY-ON-HAND
MOVE "NOT MARRIED" TO SPOUSE-NAME
```

6.6.5 Access to subtotals and totals

Access to subtotals and totals data by own code statements is only available when the totalling facility has been invoked by the user. The user may have access to the subtotals applicable at any specific key sequence break (except the minor key) or the final totals after all the data records have been read and processed.

In order to distinguish the subtotal or total of items from a given field (domain or temporary item) from the value of the previous item in that field, the operand name is prefixed by an '*' character in the own code statement, e.g.:

```
DIVIDE *STOCK-VALUE BY *QUANTITY GIVING AVERAGE-PRICE
```

The use of the '*' character is not permitted in operand-3 of an arithmetic statement nor in operand-2 of a MOVE statement.
The model problem dialogue for Stage 18 contains an example of this facility (Appendix IX Section 18).

6.6.6 Condition specification

As the user may only wish to include in the printed report data from some of his tuples, or only to execute own code statements under certain conditions, it is necessary to provide him with the means of specifying such conditions.

Condition specification is described to the user prior to the report data selection dialogue. The user may optionally remind himself about condition specification prior to the entry of his own code processing requirements.

Conditions may be simple or compound. A compound condition is made up of several simple conditions linked by the conjunctions AND or OR. The system dialogue prompts the user to enter each simple condition as a one line response and, in the case of compound conditions, to select the appropriate linking conjunction.

As parentheses are not used to indicate the hierarchy of evaluation in compound conditions (5.5), the user is reminded that after an OR conjunction it is necessary to repeat any simple condition which still applies. Each simple condition is validated and translated into a syntactically correct COBOL IF sentence by the insertion of data name qualifiers and the replacement of operators by their COBOL equivalent.

Each simple condition consists of three parts - two operands separated by an operator:

operand-1 operator operand-2

Operand-1 is the name of a temporary item or the name of the domain (field) to which the item being tested belongs.

Operand-2 specifies with what the first operand must be compared. It may be the name of a temporary item, a literal or
the name of the domain to which the compared item belongs. The text of an alphanumeric literal must be delimited by quotation marks (".").

The following four-character operators are available:

- **.EQ.** (equal to)
- **.LT.** (less than)
- **.LE.** (less than or equal to)
- **.GT.** (greater than)
- **.GE.** (greater than or equal to)
- **.NE.** (not equal)

Any of the above operators may be negated by including the word NOT immediately after the first full stop, e.g. .NOT GT. The COBOL equivalents of the above operators are shown below:

<table>
<thead>
<tr>
<th>Relational operator</th>
<th>COBOL equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EQ.</td>
<td>EQUAL TO</td>
</tr>
<tr>
<td>.LT</td>
<td>LESS THAN</td>
</tr>
<tr>
<td>.LE.</td>
<td>NOT GREATER THAN</td>
</tr>
<tr>
<td>.GT.</td>
<td>GREATER THAN</td>
</tr>
<tr>
<td>.GE.</td>
<td>NOT LESS THAN</td>
</tr>
<tr>
<td>.NE.</td>
<td>NOT EQUAL TO</td>
</tr>
<tr>
<td>.NOT EQ.</td>
<td>NOT EQUAL TO</td>
</tr>
<tr>
<td>.NOT LT.</td>
<td>NOT LESS THAN</td>
</tr>
<tr>
<td>.NOT LE.</td>
<td>GREATER THAN</td>
</tr>
<tr>
<td>.NOT GT.</td>
<td>NOT GREATER THAN</td>
</tr>
<tr>
<td>.NOT GE.</td>
<td>LESS THAN</td>
</tr>
<tr>
<td>.NOT NE.</td>
<td>EQUAL TO</td>
</tr>
</tbody>
</table>

Operand-1 and operand-2 must be of the same type, either both numeric or both alphanumeric. This is more restrictive than COBOL which permits numeric DISPLAY fields to be compared with alphanumeric fields. The restriction is imposed because numeric DISPLAY fields in the data file records are held as COMPUTATIONAL items in the Z-TUPLE group of the WORKING-STORAGE section of the generated program, and so occupy a different amount of storage space from the original field.
Stage 7 of the model problem dialogue contains an example of condition specification (Appendix IX Section 7).

6.7 SUMMARY AND APPRAISAL

The facilities of the COBOL report program generating system as proposed in the preceding sections of this chapter provide a basic but complete tool, which allows the user considerable flexibility in selecting and presenting report data. The relational view of the data and the computer dominated dialogue, with its instructional text and 'help' facility, enable the casual user to specify his report requirements without needing to understand the physical organisation of the data used.

The catalogue, containing the data base file descriptions and maintained by the data base administrator, enables access to sensitive data to be controlled. It also permits different users to have different views of the data appropriate to the problem they wish to solve. The 'flat' files described in the catalogue need not have been created exclusively for the system, but must satisfy certain criteria with regard to their organisation and contents. There is, however, provision in the design of the catalogue for the future relaxation of these criteria to enable a wider range of file types to be acceptable.

The system design is influenced by the structure and facilities of the PG/2 macro processor, but makes good utilisation of the host computer's resources by relegating non-dialogue parts of the system to background batch mode.

By the provision of new macros a number of extensions to the COBOL report generating system are possible. These would both make the present system easier to use for the casual user and provide new facilities. The latter could include the provision of
model data subfiles to enable the user to check the semantics of his own code processing and provide a more realistic sample page viewing facility. The model data subfiles would be provided by the database administrator through an extended catalogue processor.

There is also potential for extending the COBOL generating system to cover other information processing problems to include, for example, file updating.

With the increase in the number of computers in use, especially micro computers, more non-computer specialists will be called upon to use them. Research into the development of software for the generation of programs can contribute towards helping such users make efficient use of computer resources.
CHAPTER 7

IMPLEMENTATION OF SELECTED FEATURES

7.1 INTRODUCTION

The purpose of the practical work was threefold:

1. To validate aspects of the ideas presented in Chapters 5 and 6 for a dialogue based COBOL report program generator.
2. To explore the problems with dialogue creation and program generation using the PG/2 macro processor.
3. To establish the validity of the COBOL programs generated by such a system.

The volume of coding and programming effort required for the complete development of the COBOL generating system is large: probably of the same order as that required for the development of a compiler. It was not therefore possible to implement all the facilities planned for the three phases of the system.

The complete development of the first phase (the macros for maintaining the catalogue of data base file descriptions) was of prime importance, for without the catalogue there are no file descriptions on which to base the other facilities.

The design of the second and third phases of the system (the problem specifying macros and the macros for generating the complete program) is closely linked, for the former gathers the information and saves it in a form suitable for use by the latter. The problem specifying dialogue, validation of user's responses, subfile usage and formats and the macro processing for these two phases was planned in some detail, although the facilities for the wide page and default formatting facilities were omitted. Because the macros for one stage of the problem specifying dialogue make use of the information gathered in the previous stages it was most convenient to begin their development in the order in which they
are executed. The HELP macro and the macros for the first eight stages of the problem specifying dialogue were coded, although the development of macros beyond STAGE3 is incomplete.

The validity of the COBOL programs generated by the system is illustrated in Appendix X by a hand coded program to solve the problem described in Section 5.9. All stages of the program generation were carried out manually and the fabricated program compiled and executed using a specially created data base. Macro development and COBOL program testing were carried out on a teletype terminal acoustically coupled via the public telephone network to a remote ICL 1900 series computer.

The following sections of this chapter describe the implementation of the system and cover the topics shown below:

1. Outline structure of the system.
2. The use of subfiles.
3. Descriptions of the catalogue maintenance macros.
4. The use of routines within macros.
5. Descriptions of the macros for the first nine stages of the problem specifying dialogue.
6. The general strategy for processing report output format specifications.
7. Descriptions of the macros for the remaining nine stages of the problem specifying dialogue.
8. The HELP macro.
9. Generation of the complete COBOL program.

The sections are supported by material in Appendices V to X which provide additional information on subfile formats, macro processing, user response validation, COBOL generation details, samples of the problem specifying dialogue and generated COBOL program. Listings of all the macros which were coded are included in the separate folder. The chapter concludes with a summary and
Figure 7.1 Outline structure of the COBOL report program generator
7.2 OUTLINE STRUCTURE

The organisation of the COBOL report program generator, which runs under the PG/2 macro processor, falls naturally into three phases:

2. Problem specifying dialogue.
3. Generation of the complete COBOL program.

The outline structure is illustrated in Figure 7.1 and brief descriptions are given in the following subsections.

The first phase is used exclusively by the data base administrator for setting up and maintaining the catalogue of data base file descriptions.

The second phase requires interaction with the user throughout its execution, while the third phase only requires the user to initiate its execution which takes place in background mode.

The second and third phases are linked by PG/2 macro processor subfiles which are used extensively for the following purposes:

2. Macro definitions.
4. Common data.
5. 'Grown' macro definitions.
6. COBOL source program.

Further details of subfile usage are given in Section 7.3.
7.2.1 Catalogue maintenance

As described in Sections 6.3.4 and 6.3.5, the database administrator is initially provided with macro definitions for the following tasks:

1. Creating an empty catalogue.
2. Changing the catalogue password.
3. Inserting or deleting a file description.

The first two tasks are carried out by the macros CREATE and PASSCHANGE respectively. The third task is more complicated and uses four chained macros - LIBRARIAN, LIBPRELIM, LIBDELETE and LIBADD. The implementation details for these macros are given in Sections 7.6.1 to 7.6.4.

7.2.2 Problem specifying dialogue

The problem specifying dialogue is conveniently carried out in eighteen stages (6.2.2), each covering an identifiable topic and consisting of one or more macro definitions. Each macro definition automatically chains to the next. When the 'help' facility is invoked the dialogue may be resumed at the start of any of the preceding stages. Figure 7.2 illustrates the macro linkages.

The information provided by the user during the course of the dialogue is processed for use in two types of subfile. Some information is used to generate COBOL and macro-time statements which are stored in the bodies of the 'grown' macro definitions. Other information is recorded, sometimes in abbreviated form, in the common data subfiles.
Figure 7.2 Macro linkages for the stages of the problem specifying dialogue

The normal route is by the peripheral path.
7.2.3 Generation of the complete COBOL program

The complete COBOL program is synthesised by macros which generate COBOL statements from the information in the common data subfiles and make nested calls to the 'grown' macros. In this way the COBOL statements in the bodies of the 'grown' macros are regenerated to appear in their correct positions within the complete program. Further details are given in Section 7.28.

7.3 SUBFILE USAGE

The arrangement of subfiles within mainfiles is outlined and brief descriptions of each type of subfile are given in the following order:

2. Macro definitions.
4. 'Grown' macro definitions.
5. Common data subfiles.
6. COBOL source program.

The section concludes with a description of how common data subfiles are used.

Section 30 of Appendix VII contains a summary of the subfiles used by the generating system and gives details of their initialisation.

Because a limited number of subfiles may be located on a PG/2 macro processor mainfile, several mainfiles are used (4.7.3). The general strategy adopted in each macro definition is to assume that the mainfile on which the macro is located is the current mainfile. Thus any macro which needs to reference a subfile on a different mainfile must temporarily change the mainfile name and restore the original mainfile name after use. An example of this
strategy occurs in the STAGE7 macro, a listing of which is included in the separate folder (Item 26).

7.3.1 Catalogue of file descriptions

The catalogue of file descriptions is located on the CATALOGUE subfile and contains details of all the files in the database. The subfile is maintained by the database administrator and access is protected by a multilevel password system. Detailed descriptions of the format, creation and maintenance of the CATALOGUE subfile are given in Sections 6.3, 7.4, 7.5 and 7.6.

7.3.2 Macro definitions

Each stage of the problem specifying dialogue consists of one or more macro definitions. The first macro of each stage is called STAGEn, where n takes the values from 1 to 18. When a stage requires more than one macro definition, either from consideration of size or convenience, the names of the other macros take the form STAGEn-m, where m takes the values 1, 2 etc. For example, Stage 3 consists of the macros STAGE3 and STAGE3-1.

Each macro automatically chains to the next macro, but only the first macro in a multimacro stage is a check point for restarting the dialogue after a request for help (Figure 7.2). There is also the special purpose HELP macro which may be called at various points in the processing chain.

All macro definitions are stored in subfiles which have the same name as the macro.
## 7.3.3 Dialogue subfiles

These subfiles are of two types:

1. Instructional text
2. Model problem text

The instructional text which introduces each stage of the problem specification is stored in a series of subfiles, one for each stage. There is also a subfile of instructional text for the HELP macro. The use of several subfiles enables the text for one stage to be changed independently of the other stages. The use of text subfiles helps to reduce the core storage demands made by the various macros on the internal stacks and tables of the PG/2 macro processor. The instructional text subfiles are called DIAL1 to DIAL18 for the problem specifying stages and DIALHELP for the HELP macro.

There is one model problem text subfile for each stage of the problem specifying dialogue and the subfiles are called MODEL1 to MODEL18. Each subfile contains a copy of the instructional text, prompts and responses which would appear on the user's terminal at the corresponding stage during the specification of the model problem. Details of the model problem are given in Section 5.9.

In both types of dialogue subfile each line of text occupies one subfile record which may contain up to 72 characters. The contents of the appropriate text subfile is read and written out on the user's terminal, record by record, during the macros of each stage or during the execution of the HELP macro. The macro-time statements used in each macro to output the contents of a text subfile are similar. An example may be seen in the STAGE1 macro, a listing of which is included in the separate folder (Item 17).
7.3.4 'Grown' macro definitions

The statements for the 'grown' macro definitions are generated and filed in the appropriate subfile during the execution of the problem specifying stages. Each macro definition which is 'grown' is stored in a subfile of the same name as the macro and takes the form:

```
%DEF macro-name

Generated COBOL statements
and/or
generated macro-time statements.

%END
```

The body of the macro contains COBOL and/or macro-time statements which depend on the user's problem.

The generated statements are filed in the appropriate subfile using the SAVE macro-time statement. Where statements for more than one macro definition are generated within a stage, it is necessary to file the statements immediately after they are generated. If only one macro is being 'grown', the generated statements may all be filed by one SAVE macro-time statement towards the end of the stage.

The macros 'grown' during the problem specifying dialogue are executed by means of nested calls when the complete COBOL program is generated.

7.3.5 Common data subfiles

The information gathered during the problem specifying dialogue is often stored in common data subfiles. The common data subfile is referenced either in a later stage of the dialogue or during the generation of the complete COBOL program.

Common data subfiles are used because the volume of
information gathered during the dialogue greatly exceeds that which could be stored in the local, global and string variables of the PG/2 macro processor.

The format details of all the common data subfiles are given in Appendix V in the order in which they are used. An outline of how they are used is given in Section 7.3.7.

7.3.6 COBOL source program

When the complete COBOL program is synthesised the COBOL subfile is used to store the generated statements in their correct program sequence.

7.3.7 Use of common data subfiles

Throughout the stages of the problem specifying dialogue there is a progressive refinement of the information about the user's data base. Only information relevant to the current problem is retained so that subfile search time is minimised.

The catalogue of file descriptions in the CATALOGUE subfile contains details about all the data files and all fields within the records of those files in the data base. The SUBMODEL common data subfile, although similar in format to the CATALOGUE subfile, contains details of only those files and fields to which the user has password access and which are required for the generation of the COBOL program (7.10).

As the stages of the problem specifying dialogue are checkpointable a subfile written in one stage cannot be modified in a later stage. To do so would jeopardize the 'help' facility which allows the user to resume processing at an earlier stage in order to change his mind about a previous decision. Instead, data from one common data subfile must be copied to another subfile
and the latter updated. For example, the domain dictionary is first written in the DOMAINDICT subfile during Stage 3. In Stage 4 the dictionary is refined and recorded in the DOMAINLIST subfile. When, in Stage 5, the dictionary is extended to include temporary and compound domains, the DOMAININDEX subfile is used.
Figure 7.3 Outline processing for the CREATE macro

ENTRY

INTRODUCTORY TEXT

REQUEST PASSWORD

REQUEST NUMBER OF SECURITY LEVELS

VALID

SET UP AND WRITE CATALOGUE CONTROL RECORD

WRITE SUCCESS MESSAGE

ERROR MESSAGE INVALID NO. OF SECURITY LEVELS

FINISH
The CREATE macro creates the control record for an empty catalogue. The macro carries out a short dialogue with the database administrator to gather details about the catalogue password and the number of security levels by which the data files to be described therein will be protected (Figure 7.3). As the database administrator is assumed to be an experienced user the dialogue does not adopt a tutorial approach.

The text for the introductory description and the requests for information are stored in the MESSAGES subfile and are output on the user's terminal as required.

The user's response to the request for the number of security levels is validated. If it is the wrong length, non-numeric or numeric but outside the permitted range an error message is output on the user's terminal. The user is then prompted to enter a correct value.

The current date and time are obtained from the system executive and the control record is built up and written to the CATALOGUE subfile. The macro concludes by printing a message to say that the catalogue is now ready for use.

A listing of the CREATE macro is included in the separate
the separate folder (Item 11) and the dialogue from a typical run is shown together with the contents of the control record in Appendix IX (Section 19).
Figure 7.4 Outline processing for the PASSCHANGE macro

ENTRY

INTRODUCTORY MESSAGE

REQUEST CATALOGUE PASSWORD

READ CATALOGUE CONTROL RECORD

EXTRACT AND CONVERT CONTROL FIELDS

USER'S PASSWORD VALID

REQUEST NEW PASSWORD

SET UP AND WRITE CATALOGUE CONTROL RECORD

WRITE SUCCESS MESSAGE

FINISH
7.5 CHANGE THE CATALOGUE PASSWORD

The PASSCHANGE macro enables the database administrator to change the password in the catalogue control record.

The macro carries out a brief dialogue with the user to ascertain the old password. The catalogue control record is then read and the various fields extracted. If the old password has been correctly supplied the new password is requested. The current date and time are then obtained from the system executive and the catalogue control record is reconstructed to incorporate them and the new password. When the updated control record has been written on to the CATALOGUE subfile, the macro concludes by printing a message to say that the new password is effective (Figure 7.4).

An incorrectly entered old password causes the task to be abandoned after an error message has been output.

A listing of the PASSCHANGE macro is included in the separate folder (Item 12). Examples of the dialogue for an unsuccessful and a successful run are shown in Section 20 of Appendix IX.
A set of chained macro definitions, the Librarian macros, is used to carry out the insertion and/or deletions of file descriptions in the catalogue. The macros are chained because of the large number of macro-time statements required to gather and validate all the information needed to add a file description to the catalogue. In order to modify a file description it is first necessary to delete the old description and then to insert the new description.

Because the security processing for the insertion and deletion of a file have common features in the checking of passwords and file names, it is convenient to carry out the catalogue amending processes by means of four macros - LIBRARIAN, LIBPRELIM, LIBDELETE and LIBADD. Figure 6.3 illustrates the function and linkage of these macros.

The dialogue for amending the catalogue is not tutorial and prompt instructions usually occupy only one or two lines. The text is therefore located within the macro definitions instead of in a text subfile.

The following subsections give outline descriptions of each of the Librarian macros.
Figure 7.5 Outline processing for the LIBRARIAN macro
7.6.1 LIBRARIAN macro

The LIBRARIAN macro is the 'root' macro in the chain of macros for updating the CATALOGUE subfile. It prints out the introductory text and then requests the user's password to the catalogue. Only users able to enter a character string which is the same as the password in the catalogue control record are allowed to proceed further (Figure 7.5).

Global and string variables are used to pass details of file name, password, file and record counts, error codes, record pointers, etc between the chained macros. The LIBRARIAN macro extracts the file, record and security level counts from the catalogue control record and converts them to binary form for later use by the other macros. A check list of valid security levels separated by commas is also built up in string variable \#808 for use by the LIBADD macro.

A listing of the LIBRARIAN macro is included in the separate folder (item 13). Examples of both unsuccessful and successful executions of this macro are shown in Section 21 of Appendix IX.
Figure 7.6 Outline processing for the LIBPRELIM macro

ENTRY

REQUEST OPTION

ERROR MESSAGE INVALID OPTION

OTHER FINISH

ADD or DELETE

REQUEST AND VALIDATE FILENAME

ERROR MESSAGE INVALID FILENAME

VALID

Y

REQUEST FILE PASSWORD

SEARCH CATALOGUE FOR FILE NAME AND SET SUCCESS INDICATORS

LIBADD ADD OPTION DELETE LIBDELETE
7.6.2 LIBPRELIM macro

This macro definition establishes the option the user wishes to invoke and carries out the preliminary validation of file names and passwords prior to the execution of the LIBADD or LIBDELETE macro. The macro processing is illustrated in Figure 7.6 and falls into five main tasks:

1. Option dialogue
2. File name validation
3. Password dialogue
4. Catalogue search
5. Chain to the macro for the selected option

The Librarian macros enable the user to add and/or delete several file descriptions during a session at the computer terminal.

Further processing details are given in Appendix VII (Section 5). An example of the dialogue produced by the LIBPRELIM macro is included in Appendix IX (Section 22) together with samples of the LIBDELETE macro output. A listing of the LIBPRELIM macro is included in the separate folder (Item 14).
Figure 7.7 Outline processing for the LIBDELETE macro
7.6.3 LIBDELETE macro

This macro deletes a file description from the CATALOGUE subfile after verifying both its existence and the user's authority to initiate the deletion. The macro processing is illustrated in Figure 7.7 and consists of six main tasks:

1. Establish the existence of the file description.
2. Check that the user has supplied the correct password.
3. Delete the file description records from the CATALOGUE subfile.
4. Update the CATALOGUE subfile control record.
5. Output the successful deletion message.
6. Chain to the LIBPRELIM macro.

Further processing details for these tasks are given in Section 6 of Appendix VII.

The dialogue in Section 22 of Appendix IX is a sample of that produced by the LIBPRELIM and LIBDELETE macros and illustrates both unsuccessful and successful attempts to delete a file description from the CATALOGUE subfile. A listing of the LIBDELETE macro appears as Item 15 in the separate folder.
Figure 7.8 Outline processing for the LIBADD macro
7.6.4 **LIBADD macro**

The LIBADD macro carries out a dialogue with the database administrator to gather all the information required to set up and insert records for a new file description into the CATALOGUE subfile. It is by far the largest of the macros in the Librarian chain because of the volume of information which has to be requested and validated. The formats of the CATALOGUE subfile records are given in Appendix V (Section 1).

Figure 7.8 shows the processing outline for the LIBADD macro which consists of the following main tasks:

1. Establish that the catalogue does not already contain the description of a file with the same name.
2. Gather the data for the file name records and insert them in the CATALOGUE subfile.
3. Gather the data for the pairs of field description records and insert them in the CATALOGUE subfile.
4. Check that all sequence keys have been allocated.
5. Update the CATALOGUE subfile control record.
6. Output the successful insertion message.
7. Chain to the LIBPRELIM macro.

Further processing details for these tasks are given in Appendix VII (Section 7).

A listing of the LIBADD macro is included in the separate folder (Item 16). A sample of the LIBADD dialogue is shown in Appendix IX (section 23), but it does not attempt to exhaustively illustrate the error messages which invalid user responses can provoke. Also shown in the same section is a listing of the CATALOGUE after the file description has been successfully added.

Much of the validation processing for the LIBADD macro described in Appendix VII could be used as the basis for a macro to
edit an existing file description. Such a macro would, however, be as large if not larger than the LIBADD macro.

7.7 USE OF ROUTINES

As is usual in programming practice, sections of code which are common to more than one macro definition or which are executed several times within a macro, or both, are formed into routines.

Among the larger routines which appear in more than one macro are those for:

1. Checking that a user devised name obeys the rules for a COBOL data-name.

2. The validation of user specified conditions and the generation of the appropriate COBOL condition statements. Detailed descriptions of such routines are given in Appendix VII and usually follow the description of the macro in which they are first used.

Among the smaller more frequently used routines there are four which it is useful to mention at this point. The tasks they perform are as follows:

1. PROMPT routine - Issue a prompt, read the user's reply and detect a plea for help.

2. SEARCH routine - Binary search the domain dictionary for a particular domain or temporary item name.

3. INVALID RESPONSE routine - Output the 'Invalid response' message.

4. DEFAULT RESPONSE routine - Output the 'Default response' message.

Further details of these routines are given in Appendix VII (Sections 1 to 4).
Figure 7.9 Outline processing for the STAGE1 macro
The STAGE1 macro definition (Figure 7.9) is the first in the chain of macros which carry out the problem specifying dialogue. This macro performs the three tasks indicated below:

1. Sets the global variable used to contain the checkpoint number (\$CKPT) equal to 1. This variable is used to restart the COBOL generating system when the user requests help.

2. Prints the introductory text for Stage 1 which is stored in the DIAL1 subfile.

3. Chains to STAGE2, the next macro in the chain.

The DIAL1 subfile introductory text serves the following main purposes:

1. Introduces the self-teaching COBOL report program generator.

2. Informs the user how to obtain help by means of the \%HELP response and how to abandon the terminal session altogether.

3. Describes to the user how he may take advantage of the default options offered in the ensuing dialogue.

4. Introduces the concept of a relational data base and relates this to the tabular form commonly used to present data.

The computer dialogue output by the STAGE1 macro is illustrated in Appendix IX (Section 1) and a listing of the developed macro is included in the separate folder (Item 17).
Figure 7.10 Outline processing for the STAGE2 macro

ENTRY

#GCKPT=2

INTRODUCTORY TEXT FOR STAGE2

INITIALISATION

DIALOGUE TO REQUEST RELATION NAME AND PASSWORD

SEARCH RELATIONDICT FOR RELATION NAME

#GNREL = 0

Y

RELATION NAME FOUND

N

DUPLICATE RELATION NAME ERROR MESSAGE

N

#GNREL = #GNREL + 1

ENTER RELATION DETAILS IN RELATIONDICT

#GNREL=9

Z

ANY MORE RELATIONS TO ENTER

N

Y

STAGE3
The STAGE2 macro carries out a dialogue with the user in order to establish the name and password of each relation to which data access is required in order to solve the problem. The name and password details are recorded in the relation dictionary subfile, RELATIONDICT, for use by later stages of the COBOL generator.

Figure 7.10 shows the outline processing for the STAGE2 macro which consists of the following main tasks:

1. Set the checkpoint number variable (FCKPT) equal to 2. This variable is used to restart the COBOL generating system after a request for help.

2. Print the dialogue describing the use of relation names and passwords for gaining access to data. The text for the dialogue is stored in the DIAL2 subfile.

3. Prompt the user to enter up to nine relation names each with an associated password. Each unique relation name is counted and recorded together with its password in the RELATIONDICT subfile. Global variable $NREL is used to contain the number of relations in the dictionary which is maintained in ascending relation name sequence. If the user duplicates a request for a particular relation, an error message advises him that the duplicate will be ignored.

4. When the user indicates that he has no further relation
names to enter control passes to the STAGE3 macro which is the next link in the chain.

A sample of the dialogue produced by the STAGE2 macro is included in Appendix IX (Section 2) and a listing of the developed macro appears as Item 18 in the separate folder. Details of the format of the RELATIONDICT subfile are given in Appendix V (Section 2).
Figure 7.11 Outline processing for the STAGE3 macro

START

#GCKPT=3

INTRODUCTORY TEXT FOR STAGE3

INITIALISATION

READ RELATION NAME AND PASSWORD FROM RELATIONDICT

SEARCH CATALOGUE FOR RELATION DETAILS

RELATION DETAILS FOUND

Y

USE PASSWORD TO DETERMINE USER'S VIEW OF RELATION

ENTER USER'S DOMAINS IN DOMAINDICT AND SET UP KEYDICT

DISPLAY USER'S VIEW OF RELATION AND COPY DETAILS TO SUBMODEL

N

ALL RELATIONS PROCESSED

ERROR INDICATOR SET

ERROR MESSAGE AND RECOVERY OPTION DIALOGUE

ERROR INDICATOR SET

Y

STAGE3-1

ABANDON RUN

Y

FINISH

N

STAGE2

START
In the third stage of the problem specifying dialogue the user is shown details of the domains belonging to the relations whose names he entered in Stage 2 and to which he has password access.

The dialogue text for this stage is in the DIAL3 subfile. The contents of the RELATIONDICT subfile is used to determine which information in the CATALOGUE subfile should be displayed to the user.

During this stage data is recorded in three subfiles for later use by the generating system. These subfiles are called DOMAINDICT, KEYDICT and SUBMODEL and contain, respectively, details of all the domains to which the user may have access, the names of
Figure 7.12 Outline processing for the STAGE3-1 macro
the sequence key domains of the user's relations and the subset of catalogue file descriptions to which the user needs access. This processing is illustrated in Figure 7.11.

In addition to displaying details of the user's data base submodel, the Stage 3 processing verifies that the relations selected by the user are not disjoint or disconnected. This means that it will be possible to generate the COBOL file handling statements necessary to match up records from all the specified files. Figure 7.12 outlines this verification process.

The Stage 3 processing is carried out by two macro definitions, STAGE3 and STAGE3-1, the former automatically chaining to the latter on successful completion. It is possible that in subsequent versions of the COBOL generator that the criteria for defining disjoint relations may be relaxed (6.4.2), hence the decision to carry out this aspect of the Stage 3 processing in the separate STAGE3-1 macro.

The data for the SUBMODEL and DOMAININDICT subfiles is recorded during the STAGE3 macro processing. In this macro the KEYDICT subfile is used only as temporary storage while the information about each relation is processed. When more than one relation is required for the problem, the permanent data for the KEYDICT subfile is recorded during the processing of the STAGE3-1 macro.

Format details for the three subfiles written during the third stage are given in Appendix V (Sections 3 to 5).

An example of the Stage 3 dialogue is given in Appendix IX (Section 3).

Further details of the processing in the STAGE3 and STAGE3-1 macros are given in Appendix VII (Sections 8 and 9) where future enhancements are also considered. Listings of these macros are included in the separate folder, although STAGE3-1 is not fully developed (Items 19 and 20).
Figure 7.13 Outline processing for the STAGE4 macro
During this stage the user is reminded of all the domains which are in the data base as viewed by him (Figure 7.13). As each domain name is displayed the user is invited to indicate whether or not it is required to solve the current problem. If the domain is required and it contains numeric data, the user is invited to avail himself of the automatic totalling facility. This facility will automatically accumulate the total of all the items in the domain which are retrieved from the data base.

The STAGE4 macro uses the DIAL4 and DOMAINDICT subfiles as input and outputs data to the DOMAINLIST subfile. The DOMAINLIST subfile is similar to DOMAINDICT, but contains details of only those domains required to solve the problem. Each domain record, however, contains an extra field to indicate whether or not totals for it will have to be accumulated. Details of the record formats used in the DOMAINLIST subfile are given in Appendix V (Section 6).

An example of the Stage 4 dialogue is included in Appendix IX (Section 4). Processing details for the STAGE4 macro are given in Section 10 of Appendix VII and a listing of the partially developed macro is included in the separate folder (Item 21).

The design of the Stage 4 dialogue was greatly influenced by
the fact that the input/output device used for development was a teletype. If a more powerful device such as a visual display device unit was available, a dialogue adopting a 'menu' facility for the selection of problem data would be more appropriate.
Figure 7.14 Outline processing for the STAGE5 macro

ENTRY

#GCKPT=5

INTRODUCTORY TEXT FOR STAGES

COPY DOMAIN RECORDS FROM DOMAINLIST TO DOMAININDEX

GENERATE INITIAL STATEMENTS FOR TEMPITEM MACRO

CREATE TEMPORARY ITEMS

STAGES-2

REQUEST AND VALIDATE ITEM NAME

SEARCH DOMAININDEX FOR ITEM NAME

ERROR MESSAGE DUPLICATE NAME

REQUEST AND VALIDATE ITEM SIZE

ENTER INITIAL VALUE

INITIAL VALUE SET TO BLANKS

REQUEST AND VALIDATE INITIAL VALUE

ENTER ITEM DESCRIPTION IN DOMAINDOMAINS

GENERATE COBOL DESCRIPTION FOR TEMPITEM SUBFILE
This stage allows the user to extend his data base by the creation of temporary items or domains which are available only to the COBOL program currently being generated. The temporary data belongs to one of three categories:

1. Alphanumeric
2. Numeric
3. Compound alphanumeric

For convenience the Stage 5 processing is carried out by three macro definitions, STAGE5, STAGE5-1 and STAGE5-2, one for each category.

Apart from the totalling facility for numeric items, the setting of initial values and the use of temporary data is entirely under user control. Initial values are specified in this stage and the processing is specified in Stage 18.

The Stage 5 processing uses subfiles DIAL5 and DOMAINLIST as input and outputs data to three subfiles, DOMAININDEX, TEMPITEM
Figure 7.15 Outline processing for the STAGE5-1 macro
and CONCAT. The DOMAININDEX subfile is similar to DOMAINLIST, but it is without the initial record and also contains information about the temporary and compound domains created by the user. The format details for the DOMAININDEX subfiles are given in Appendix V (Section 7).

The count of DOMAININDEX records is maintained in global variable #GDMCT so that it can readily be updated by any of the three macros. TEMPITEM and CONCAT are macro definition subfiles, the statements for which are generated during this stage.

The TEMPITEM macro is used during the synthesis of the generated COBOL program to insert data description statements for temporary data, if any. The macro takes the form:

```cobol
%DEF TEMPITEM
% LABELOFF
COBOL data description statements for temporary and compound items
% LABELON
%END
```

The data description statements may be generated in any of the three Stage 5 macros from the user supplied information.

The CONCAT macro is used during program synthesis to insert MOVE statements which set up the data in the compound domains, if any. The macro takes the form:

```cobol
%DEF CONCAT
% LABELOFF
COBOL MOVE statements to set up data in compound domains
% LABELON
%END
```

The MOVE statements are generated in the STAGE5-2 macro from the information about compound domains supplied by the user.

A useful enhancement to the Stage 5 macros would be a data summary. A list of all the temporary and compound domains created during the stage could be output on the user's terminal. There is also potential for extending the HELP macro facilities to allow the user to inspect, possibly selectively, details of all the
Figure 7.16 Outline processing for the STAGE5-2 macro
domains in his submodel data base.

The outline processing for the Stage 5 macros is illustrated in Figures 7.14 to 7.16 and further processing details are given in Appendix VII (Sections 11 to 13). An example of the Stage 5 dialogue is included in Appendix IX (section 5) and listings of the partially developed macros are included in the separate folder (Items 22 to 24).
Figure 7.17 Outline processing for the STAGE6 macro

ENTRY

#GCKPT=6

*GREL?

N

INTRODUCTORY TEXT FOR STAGE6

REQUEST AND VALIDATE MISMATCH OPTION

MESSAGE REQUIRED

N

SPECIFY OWN MESSAGE

N

REQUEST AND READ USER'S MESSAGE

STAGE7
The user is only aware of the existence of the Stage 6 processing if more than one relation (file) is required to solve his problem.

Because the data files used by the generated COBOL program are created and maintained by other programs there may be a lack of integrity in the user's data base submodel. The user is offered a choice between three ways of handling any inconsistency encountered by the COBOL program when matching record keys from two or more data files:

1. The unmatched records may be ignored.

2. A dummy matching record may be created with zeros in the numeric non-key fields and blanks in the alphanumeric non-key fields.

3. The run may be terminated.

In all cases the user is given the option of having an error message output by the program (Figure 7.17). The error message will give details of the relation name (file name) and the key value together with a brief comment, the text of which may be specified by the user.

The STAGE6 macro uses the descriptive text stored in the DIAL6 subfile. The user's options are stored in global and string variables for use in a later stage.

A sample of the Stage 6 dialogue is included in Appendix IX (Section 6) and a listing of the partially developed macro is included in the separate folder (Item 25). Further processing details for the STAGE6 macro are given in Appendix VII (Section 14).
Figure 7.18 Outline processing for the STAGE7 macro

ENTRY

#GCKPT=7

GENERATE INITIAL STATEMENTS FOR SECOND MACRO

DATA SELECTION

INSTRUCTIONAL TEXT FOR STAGE7

CONDITION SPECIFYING ROUTINE WITH COBOL GENERATION

COMPLETE GENERATION OF COBOL IF SENTENCE

GENERATE FINAL STATEMENTS FOR SECOND MACRO

STAGE8
The STAGE 7 macro dialogue finds out whether all the data retrieved from the database should be included in the report or if only a selection is required. In the latter case the dialogue explains, with examples, the form of a simple condition and how several may be linked to form a compound condition. The user is then prompted to specify the selection condition applicable to his problem. When a compound condition is used it is entered as two or more simple conditions and the user is prompted to specify the appropriate conjunctions.

The instructional text for the STAGE7 macro is stored in the DIAL7 subfile and the macro also reads data from the DOMAININDEX subfile during the validation of the user's simple conditions. The SELCOND macro statements are generated at this stage. During the synthesis of the generated program the SELCOND macro is used to insert the COBOL conditional statements, if any, before those which process the retrieved data. The macro takes the form:
As the statements in the own code facility offered during Stage 18 may also be conditional, the condition specifying dialogue and validation of the user's responses are carried out by routines which are similar in both stages. The outline processing for the STAGE7 macro is illustrated in Figure 7.18. Further processing details for the STAGE7 macro and the condition specifying routines are given in Appendix VII (Section 15).

A sample of the Stage 7 dialogue is given in Appendix IX (Section 7) and a listing of the partially developed STAGE7 macro is included in the separate folder (Item 26).
Figure 7.19 Outline processing for the STAGE8 macro

ENTRY

GCKPT=8

INTRODUCTORY TEXT FOR STAGE8

PAGE WIDTH, DEPTH AND FORMAT DIALOGUE

INITIALISATION FOR LABEL ASSIGNMENT

LABEL ASSIGNMENT INSTRUCTIONAL TEXT

DISPLAY DOMAIN NAME AND REQUEST LABEL

DOMAIN IN REPORT

VALIDATE LABEL AND WRITE LABEL TABLE RECORD

ALL DOMAINS FOR ALL LABELS

CLEAR UNUSED LABELS, IF ANY
IN LABEL TABLE SUBFILE

FURTHER INSTRUCTIONAL TEXT ABOUT REPORT FORMATS

STAGE9
In this stage the user is invited to select the dimensions of his report page and is introduced to the notation used for specifying report line formats.

In order to specify report line formats a label character must be assigned to each domain or temporary item which appears in the report. The label character is used to denote the print positions to be occupied by the item. As there are nineteen possible label characters up to nineteen domain or temporary items may be included in the report (6.5.8).

The user is reminded of all the domains and temporary items which feature in his problem, and is invited to assign the label character of his choice to those which will appear in the printed report (Figure 7.19).

The instructional text for the STAGE8 macro dialogue is input from the DIAL8 subfile and details of the user's problem domains and temporary items are read from the DOMAININDEX subfile.

During the execution of the STAGE8 macro, details of the domain or temporary item associated with each label character used are recorded in the LABELTABLE subfile. In the records of unused labels all fields except the first are empty. The format of
LABELTABLE subfile records is shown in Appendix V (Section 8).

A sample of the Stage 8 dialogue is included in Appendix IX (Section 8) and a listing of the partially developed macro is included in the separate folder (Item 27). Processing details for the STAGE8 macro appear in Appendix VII (section 16).
Figure 7.20 Outline processing for the STAGE9 macro
In Stage 9, the user is introduced to the editing facilities available when alphanumeric or numeric items are printed in the report. The user is invited to experiment by designing editing formats, both alphanumeric and numeric, with an opportunity for seeing the result of editing data values of his choice. Figure 7.20 illustrates the outline processing for the STAGE9 macro.

The STAGE9 macro is normally entered by chaining from the STAGE8 macro. It may, however, be entered from the HELP macro, where experimental editing is one of the facilities offered to users who have passed beyond the eighth stage of the problem specifying dialogue. (Figure 7.2). On completion, the STAGE9 macro passes control either to the STAGE10 macro or to the macro from which the plea for help was made. The value of the checkpoint variable #GCKPT is used to determine the next macro in the chain.

The instructional text for the STAGE9 macro is stored in subfile DIAL9. This text is suppressed when the macro is entered via the help facility as the user will already have seen it.

The processing details for the STAGE9 macro are summarised in note form in Appendix VII (Section 17). The Stage 9 instructional text is illustrated in Appendix IX (Section 9).
SPECIFICATIONS

The dialogue in which the formats of the different report output categories are specified is similar. However, it is more convenient to process each category of output in a separate stage for the following reasons:

1. In each category different field types are permitted.
2. There are variations in the COBOL Procedure division statements generated for each category.
3. The dialogue for any output category may be repeated after viewing a sample page in Stage 17, but they are not repeated in the original order.
4. The combined processing for all categories of output would result in a complicated macro too large for the PG/2 macro processor's internal stacks and tables.

In each stage in which the user opts to specify output formats statements for two macro definition subfiles are generated. One subfile has the name suffix REC and the other the name suffix PARA. Information is also written to a common data subfile which has the name prefix PAGE.

The body of the macro with the name suffix REC contains COBOL group descriptions for the non-blank records of the output category. The general form of the statements generated for the macro is as follows:

```cobol
%DEF .....REC
% LABELOFF
  01 line-name-1.
    02 ... { Field descriptions for 1st line. }
    02 ... { 1st non-blank record description. }
    ... { 2nd non-blank record description. }
  01 line-name-2.
    02 ... { Field descriptions for 2nd line. }
    02 ... % LABELON
%END
```
These record descriptions are destined for the Working-storage section of the COBOL program.

The body of the macro with the name suffix PARA contains the Procedure division statements which write out the non-blank lines of the output category.

For each valid line format specified by the user, a record is written to the subfile with the name prefix PAGE. In Stage 17 the contents of this subfile is used if the user wishes to view a sample page. All common data subfiles with the name prefix PAGE have the same format as that shown in Section 9 of Appendix V.

For each valid non-blank line format entered by the user statements are generated for both the REC and PARA name suffix macro definition subfiles. The statements generated for the former are saved in the subfile before the statements for the latter are generated.

The level O2 data description statements are generated as each field in the line format specification is identified and validated, but they are not filed until all the fields in the line have been processed. If an error is detected during the validation, the statements for the previous fields in the line, if any, are emptied on to the WORK subfile. This serves as a convenient 'sink' in which to empty the PG/2 output stack so that it is 'cleaned up' ready to process the user's corrected line format specification. A meaningful error message is output on the user's terminal and he is prompted to type in the amended line format specification.
The STAGE10 macro carries out the dialogue in which the user specifies the formats of the detail lines which are to be printed for each retrieval from his data base. The outline processing for the STAGE10 macro is shown in Figure 7.21.

The macro uses two input subfiles, DIAL10 and LABELTABLE, and generates statements for macro definition subfiles DETAILREC and WRITEPARA. It also writes information on the common data subfile PAGEDETAIL. The WORK subfile is used only when a format specification error is detected.

A sample of the Stage 10 dialogue is shown in section 10 of Appendix IX.
Figure 7.21 Outline processing for the STAGE10 macro
7.18.1 STAGE10 macro processing notes

1. The STAGE10 macro is normally entered by chaining from the STAGE9 macro, but may also be entered from the STAGE17-3 macro.

2. Checkpoint variable $GCKPT is used to determine the method of entry and, on completion, the next macro in the chain.

3. Subfile DIAL10 contains the text which introduces the stage and describes the use of text literals. The DIAL10 text is output only if STAGE10 is entered normally.

4. The LABELTABLE subfile (Appendix V Section 8) is used during the validation of line format specifications.

5. The level 01 line-names used in the body of the DETAILREC macro are Z-DETAIL-1, Z-DETAIL-2 etc.

6. Particulars of the identification, validation and generation of the level 02 field descriptions for the four field types permitted in a detail line are given at the references indicated below:

   Blank FILLER (Appendix VII Section 18.1)
   Alphanumeric literal FILLER (Appendix VII Section 18.2)
   Alphanumeric data item (Appendix VII Section 18.3)
   Numeric data item (Appendix VII Section 18.4)

7. The COBOL statements generated for the body of the WRITEPARA macro will ultimately be located in the Z-PARA-WRITE paragraph. This paragraph is performed once for each tuple of data retrieved (See Appendix VII Section 19 for macro details).

8. Format details for the PAGEDETAIL subfile are given in Appendix V (Section 9).
Figure 7.22 Outline processing for the STAGE11 macro

- ENTRY
- \#GCKPT = 11
- INSTRUCTIONAL TEXT FOR STAGE11
- STAGE12
In this stage the user is told of the extra data items, date, time and page number, which he may include in certain lines of his printed report. The text explaining which strings of characters should be used to denote the print positions of these extra data items is stored in subfile DIAL11.

The processing in the STAGE11 macro, illustrated in Figure 7.22, consists of only three steps:

1. Set the checkpoint variable equal to 11.
2. Print the instructional text contained in the DIAL11 subfile.
3. Chain to the STAGE12 macro.

There are three main reasons for making the output of this instructional text into a separate stage, rather than incorporating it at the beginning of the next stage. First it reduces the volume of text displayed when the user restarts the report title specification after a plea for help. Secondly, there is potential for increasing the number of extra items accessible to the user, e.g. the line counter. Thirdly, users may find it desirable to be offered the use of these items earlier in the dialogue so that they may be included in detail line specifications. As a separate stage the order of the problem specification dialogue is more easily changed.

The instructional text for Stage 11 is illustrated in Section 11 of Appendix IX.
The STAGE12 macro carries out the dialogue in which the user specifies the format of the report title, if required. The output subfiles shown in the above diagram are used only if a report title is specified.

A sample of the Stage 12 dialogue is shown in Section 12 of Appendix IX and the outline processing is illustrated in Figure 7.23.
Figure 7.23 Outline processing for the STAGE12 macro

ENTRY

\#GCKPT = 11

\#GCKPT = 12

INTRODUCTORY TEXT FOR STAGE12

TITLE REQUIRED

\#GTITL = 0

\#GTITL = 1

DIALOGUE AND RESPONSE PROCESSING FOR TITLE SPECIFICATION

\#GCKPT = 12

STAGE17-1

STAGE13
7.20.1 STAGE12 macro processing notes

1. The STAGE12 macro is normally entered by chaining from the STAGE11 macro, but may also be entered from the STAGE17 macro.

2. Checkpoint variable #GCKPT is used to determine the method of entry and, on completion, the next macro in the chain.

3. The introductory text for the stage is output from the DIAL12 subfile, but only if the macro is entered normally.

4. Global variable #GTITL is used to indicate whether or not a report title is specified.

5. The line-names generated for the level 01 statements in the body of the TITLEREC macro are Z-TITLE-1, Z-TITLE-2 etc.

6. Particulars of the identification, validation and generation of the level 02 field descriptions for the five field types permitted in a title line are given at the references indicated below:
   - Blank FILLER (Appendix VII Section 18.1)
   - Alphanumeric literal FILLER (Appendix VII Section 18.2)
   - Date item (Appendix VII Section 18.5)
   - Time item (Appendix VII Section 18.6)
   - Page number item (Appendix VII Section 18.7)

7. The body of the TITLEPARA macro consists of COBOL Procedure division statements for the Z-PARA-TITLE paragraph which writes out the report title page. This paragraph is performed during the initial section of the COBOL program. The general form of the statements generated for the TITLEPARA macro is shown in Appendix VII (Section 20).

8. Format details for the PAGETITLE common data subfile are given in Appendix V (Section 9). Blank lines after the last non-blank title line are not included in the subfile.
The STAGE13 macro carries out the dialogue with the user in which he may, if he desires, specify a page heading to appear at the top of each report page after the title page, if present.

Since the page heading is optional not all the subfiles shown in the above diagram may be used during the execution of the macro.

A sample of the Stage 13 dialogue is shown in Section 13 of Appendix IX and the outline processing is illustrated in Figure 7.24.
Figure 7.24 Outline processing for the STAGE13 macro
7.21.1 STAGE13 macro processing notes

1. The normal method of entry is from the STAGE12 macro, but entry may also be from the STAGE17-1 macro.

2. #GCKPT, the checkpoint variable, is used to determine the method of entry and, on completion, the next macro in the chain.

3. The introductory text which describes the page heading facility is stored in the DIAL13 subfile. The full text is output only if sequence break headings may follow the page heading (i.e. if #GNKEY > 1). The introductory text is not printed if the macro is entered from the STAGE17-1 macro.

4. Global variable #GPAGE is used to indicate whether or not a page heading is specified.

5. The LABELTABLE subfile (Appendix V Section 8) is used during the validation of line format specifications, but only if they contain label characters.

6. Statements for the PHEADREC macro are generated only if a page heading is specified. Z-PAGE-1, Z-PAGE-2, etc are used as the level 01 line-names in the macro body.

7. All seven field types are permitted in a page heading line. Particulars of the identification, validation and level 02 statement generation are given in Appendix VII (Section 18).

8. The PHEADPARA macro is always generated. Its body contains COBOL statements destined for the Z-PARA-PAGE paragraph (See Appendix VII Section 21 for macro details).

9. The PAGEHEAD subfile is recorded only if a page heading is specified. The subfile format details are given in Section 9 of Appendix V.
If the user's data is sequenced on more than one key domain, the STAGE14 macro carries out the dialogue which invites the user to specify sequence break headings for his report output. Sequence break headings may be specified for each key level except the minor key. Only if the option is exercised are the output subfiles used. Separate output subfiles are used for each key level which is denoted by n in the above diagram and in the following text.

A sample of the Stage 14 dialogue is included in Section 14 of Appendix IX and the outline processing is illustrated in Figure 7.25.
Figure 7.25 Outline processing for the STAGE14 macro

ENTRY

#GNKEY > 1

#GCKPT = 13

#GCKPT = 14

INTRODUCTORY TEXT FOR STAGE14

SUBHEADINGS REQUIRED

#GHEAD = 0

#GHEAD = 1

#GSBHD = 1

DIAGRAM AND RESPONSE PROCESSING FOR SEQUENCE BREAK HEADINGS AT KEY LEVEL. #GSBHD

#GCKPT = 14

#GSBHD = #GSBHD + 1

STAGE17-2

STAGE15
7.22.1 STAGE14 macro notes

1. The macro is normally entered from STAGE13, but may also be entered from STAGE17-2.

2. Global variable $GCKPT is used to determine the method of entry and hence, on completion, the next macro in the chain.

3. Global variable $GHEAD is used to indicate whether or not sequence break headings have been specified.

4. When entered normally, sequence break headings are specified in major to penultimate minor key order. The current key level is indicated by $GSBHD. When the alternative method of entry is used a single value for $GSBHD is set in the STAGE17-2 macro.

5. The introductory text for the stage is output from DIAL14 only if the macro is entered normally.

6. The KEYDICT subfile (Appendix V Section 5) is used to associate a key name with the current key level for dialogue purposes.

7. The line-names used in level 01 statements generated for the SBHRECn macro are Z-HEADn-1, Z-HEADn-2 etc, where n is the current key level.

8. The LABELTABLE subfile (Appendix V Section 8) is used in the validation of the line specification formats which may include only field types described in Sections 18.1 to 18.4 of Appendix VII.

9. The COBOL statements generated in the body of the SBHPARAn macro are destined for the Z-HEADn-WRITE paragraph. (See Appendix VII Section 22 for macro details.)

10. Format details for the PAGESBHN common data subfiles are given in Appendix V (Section 9).
Only if the user availed himself of the totaling facility offered in Stages 4 and 5, i.e. if \( # \text{TOT} = 1 \), and the data is sequenced on more than one key, i.e. \( # \text{NKEY} > 1 \), does the STAGE15 macro invite the user to specify formats for subtotal lines. Subtotal line formats may be specified for each key level except the minor key. Output subfiles are used only if the option is exercised, in which case separate subfiles are used for each key level. The key level is denoted by \( n \) in the above diagram and in the following text.

The KEYDICT and LABELTABLE subfiles (Appendix V Sections 5 and 8) are used in the same way as in the STAGE14 macro. Section 15 of Appendix IX contains a sample of the Stage 15 dialogue and Figure 7.26 illustrates the outline processing.
Figure 7.26 Outline processing for the STAGE15 macro

ENTRY

#GOTO = 1

Y

#GNKEY > 1

Y

#GCKPT = 14

Y

#GCKPT = 15

INTRODUCTORY TEXT FOR STAGE15

SUBTOTAL LINES REQUIRED

N

#GSUBT = 0

Y

#GSUBT = 1

#GSBTL = #GNKEY - 1

Y

DIAGNOSIS AND RESPONSE PROCESSING FOR SUBTOTAL LINES AT KEY LEVEL #GSBTL

#GCKPT = 15

N

STAGE17-4

Y

#GSBTL = #GSBTL - 1

N

STAGE16

#GSBTL ≤ 0

Y

#GSBTL = #GSBTL - 1

N

STAGE16
STAGE15 macro notes

1. The macro is normally entered by chaining from the STAGE14 macro, but may also be entered from macro STAGE17-4.

2. The method of entry and, on completion, the next macro in the chain is determined from global variable GCKPT.

3. Global variable GSUBT is used to indicate whether or not subtotal lines are specified.

4. When entered normally subtotal lines are specified in penultimate minor to major key order. Global variable GSBTL is used to indicate the current key level. When the alternative method of entry is used a single value for GSBTL is set in STAGE17-4.

5. The introductory text from the DIAL15 subfile is only output if the macro is entered normally.

6. The line-names used in the level 01 statements generated for the STLRECn macro are Z-SUBTOTALn-1, Z-SUBTOTALn-2, etc.

7. Only the field types described in Sections 18.1 to 18.4 of Appendix VII may be included in subtotal line format specifications.

8. The COBOL statements generated in the body of the STLPARAn macro are destined for the Z-TOTALn-WRITE paragraph. (See Section 23 of Appendix VII for macro details.)

9. Format details for the PAGESTLn common data subfiles are given in Appendix V (Section 9).
Only if the user took advantage of the totalling facility offered in Stages 4 and 5, i.e. if global variable #GTOT is equal to 1, does the STAGE16 macro invite the user to specify line formats for total lines.

The output subfiles are written only if total lines are specified for the report.

A sample of the Stage 16 dialogue is included in Section 16 of Appendix IX and the outline processing is illustrated in Figure 7.27.
Figure 7.27 Outline processing for the STAGE16 macro

ENTRY

#G_WIDTH=1

#GCKPT = 15

#GCKPT=16

INTRODUCTORY TEXT FOR STAGE16

TOTAL LINES REQUIRED

#G_WIDTH=0

#G_WIDTH=1

DIALOGUE AND RESPONSE PROCESSING FOR TOTAL LINES SPECIFICATION

STAGE17
7.24.1 STAGE16 macro notes

1. The STAGE16 macro is entered by chaining from the STAGE15 macro or from the STAGE17-5 macro.

2. Checkpoint variable \texttt{GCKPT} is used to determine the method of entry. Only when entry is from STAGE15 is the introductory text from the DIAL16 subfile output to the user.

3. Global variable \texttt{GTOTL} is used to record whether or not total lines are specified.

4. The line-names used in the level 01 statements generated for the TOTLREC macro are Z-TOTAL-1, Z-TOTAL-2, etc.

5. The LABELTABLE subfile (Appendix V Section 8) is used during the validation of line format specifications only when they contain label characters. Only the field types described in Appendix VII Sections 18.1 to 18.4 may be used.

6. The COBOL statements generated for the body of the TOTL PARA macro are destined for the Z-TOTALO-WRITE paragraph. (See Section 24 of Appendix VII for macro details.)

7. Format details for the PAGETOTL subfile are given in Appendix V (Section 9).
The purpose of the Stage 17 problem specifying dialogue is to give the user the opportunity to view a stylized sample of a report page based on the format specifications which he entered during the earlier stages of the dialogue. Should the composite page prove displeasing or if the user's design cannot be accommodated within the number of lines chosen as the maximum for a report page, the user is invited to redesign the formats for the report output categories.

In order to be able to enter and return from the format specifying macros of the previous stages, six macro definitions are required for Stage 17. The Stage 17 macro linkages originally
Figure 7.26 Macro linkages for Stage 17
illustrated in Figure 7.2 are redrawn in Figure 7.28 with the linkages to the HELP macro omitted. Figures 7.29 to 7.34 and the accompanying notes outline the processing carried out by the six Stage 17 macros.

In addition to the DIAL17 text subfile and the key dictionary subfile KEYDICT, all the subfiles with the PAGE name prefix, written in Stages 10, 12, 13, 14, 15 and 16, are used when building up the sample report pages.

A stylized report page is included in the sample of Stage 17 dialogue which appears in Section 17 of Appendix IX.
Figure 7.29 Outline processing for the STAGE17 macro

ENTRY

#GCKPT=17

INTRODUCTORY
TEXT FOR
STAGE17

OVERSIZE
PAGES(S)
SPECIFIED

Y

N

PAGE
VIEWING
TEXT

VIEW
PAGE
SAMPLE(S)

Y

N

DISPLAY
PAGE
SAMPLE(S)

PAGE
DESIGN
SATISFACTION

Y

N

#GSBHD=0
#GSBT=0
#CNKEY

REDESIGN
REPORT
TITLE

Y

N

STAGE18

STAGE12

STAGE17-1
7.25.1 STAGE17 macro notes

1. If any of the conditions set out in Section 25 of Appendix VII for the combinations of report output categories to be accommodated on a report page are not satisfied, an oversize page is deemed to have been specified. In this case a meaningful error message is output on the user's terminal and he is asked to redesign the page layout.

2. The data in the subfiles with the name prefix PAGE (Appendix V Section 9) is used to form the page samples displayed on the user's terminal. The title page, if specified, consists of the line records from the PAGETITLE subfile and sufficient blank lines to make up a full length page. The sample report page consists of one copy of the line records for each category of output specified by the user. The line records for the detail category are, if space permits, repeated several times to fill out the sample page to near maximum length. The output categories appear in their natural order, i.e. page heading, sequence break headings, detail lines, sequence break subtotals and totals. When the page length is too short to accommodate subtotal and total categories these are made to appear on a separate headed page or pages. In all cases where line records from the PAGE name subfiles begin with the greater than character (>), this is replaced by a blank before the line is output for the user.

3. Global variables $GSBHD and $GSBTL are initially set equal to zero and $GKEY respectively. These variables are used to indicate the current key level when sequence break headings and subtotals output formats are respecified (7.22 and 7.23).

4. The first report output category which the user is invited to redesign is the report title.
Figure 7.30 Outline processing for the STAGE17-1 macro

ENTRY

REDRAW PAGE HEADING

STAGE13

STAGE17-2

Figure 7.31 Outline processing for the STAGE17-2 macro
7.25.2 STAGE17-1 macro notes

The STAGE17-1 macro asks the user to specify whether or not he wishes to redesign the page heading. If the reply is 'yes' control is passed to the STAGE13 macro, otherwise control passes to the STAGE17-2 macro.

7.25.3 STAGE17-2 macro notes

1. If the number of sequence keys in the user's data ($\#\text{NKEY}$) is greater than 1 and the user had previously specified sequence break headings, i.e. $\#\text{HEAD}$ equals 1, he is given the opportunity to redesign their layouts. Otherwise control passes to the STAGE17-3 macro.

2. The user is invited to redesign the sequence break heading for each key domain in major to penultimate minor key order. Global variable $\#\text{GSBHD}$ is used to denote the key level of the current sequence break heading. This variable is initialised to zero in the STAGE17 macro and is incremented in the dialogue processing loop of this macro. $\#\text{GSBHD}$ is also used to look up the name of the current key domain in the KEYDICT subfile. Control passes to the STAGE14 macro if the user accepts the invitation to redesign a specific sequence break heading.

3. When all sequence break headings have been offered to the user, i.e. $\#\text{GSBHD}$ equals $\#\text{NKEY}$, control passes to the STAGE17-3 macro.
Figure 7.32 Outline processing for the STAGE17-3 macro

ENTRY

REDRAW DETAIL LINES

Y

STAGE10

N

STAGE17-4

Figure 7.33 Outline processing for the STAGE17-4 macro

ENTRY

#GTOT=1

N

Y

#GKEY >1

N

Y

#GSUBT=1

N

Y

#GSBTL =
#GSBTL -1

#GSBTL =0

Y

STAGE17-5

N

REDESIGN SUBTOTALS AT LEVEL #GSBTL

Y

STAGE15

N
7.25.4 STAGE17-3 macro notes

The STAGE17-3 macro asks the user if he wishes to redesign his detail line formats. If the reply is 'yes', control passes to the STAGE10 macro, otherwise control is passed to the STAGE17-4 macro.

7.25.5 STAGE17-4 macro notes

1. If the following three conditions are satisfied the user is given the opportunity to redesign the sequence break subtotal layouts, otherwise control passes directly to the STAGE17-5 macro:
   - Totalling facility invoked ($^\text{TOT} = 1$).
   - There is more than one sequence key in the user's data ($^\text{NKEY} > 1$).
   - Sequence break subtotals have previously been specified ($^\text{SUBT} = 1$).

2. The user is invited to redesign the sequence break subtotal output for each key domain in penultimate minor to major key order. Global variable $^\text{GSBTL}$ is used to denote the key level of the current sequence break output. This variable, initialised to the value of $^\text{NKEY}$ in the STAGE17 macro, is decremented in the dialogue processing loop. It is also used to look up the name of the current key domain in the KEYDICT subfile. Control passes to the STAGE15 macro if the user accepts the invitation to redesign the subtotal output for a sequence break on a specific key.

3. When all the sequence break subtotal outputs have been offered, i.e. $^\text{GSBTL}$ equals 1, control passes to the STAGE17-5 macro.
7.25.6 STAGE17-5 macro notes

If the totalling facility was invoked, i.e. \#GTOT is equal to 1, the STAGE17-5 macro invites the user to redesign his totals output. If he accepts the offer control passes to the STAGE16 macro, otherwise it is passed to the STAGE17 macro where the validity on any layout changes can be assessed.
The major function of the STAGE18 macro is to explain to the user the own code processing facilities and, if required, to carry out the dialogue in which he specifies his requirements. Because they require relatively few macro-time statements the following three subsidiary tasks are included in order to avoid yet another stage.
1. Invite the user to enter comments on the problem specifying stages.

2. Generate statements for the CONSTANTS macro definition subfile which is used during the generation of the complete COBOL program.

3. Instruct the user how to exit from the PG/2 macro processor.

The instructional text located in the DIAL18 subfile describes the use of the five own code statements ADD, SUBTRACT, DIVIDE, MULTIPLY and MOVE (6.6.4). As the execution of each own code statement may be conditional some of the text in the DIAL7 subfile may be used if the user wishes to be reminded of how to specify conditions.

For each point in the COBOL report program where own code processing may be inserted there exists a macro definition subfile with OWN as the name prefix. Further details of the subfile associated with each processing point is given in Appendix VII (Section 26). The statements for the macros at the selected own code processing points are generated during this stage. The body of each macro contains the COBOL statements generated from the information supplied in the user's own code specifications.

The DOMAININDEX subfile is referenced during the validation of the user's own code and condition statements, if any.

The COMMENTS subfile is a text subfile to which any comments the user may choose to enter about the COBOL generating system are appended. The subfile contents are available to those maintaining the COBOL generating system.

The CONSTANTS macro definition subfile is used to pass values of string and global variables between the problem specifying phase and the COBOL program generating phase.

A sample of the Stage 18 dialogue is included in Section 18
Figure 7.35 Outline processing for the STAGE18 macro

ENTRY

GCKPT=18

INTRODUCTORY TEXT FOR STAGE18

USE OWN CODE

Y

INSTRUCTIONAL TEXT ABOUT OWN CODE OPERATIONS

N

INSTRUCTIONAL TEXT ABOUT OWN CODE STATEMENTS CARRIED OUT CONDITIONALLY

Y

CONDITION SPECIFICATION REMINDER

N

INSTRUCTIONAL TEXT ABOUT SPECIFYING CONDITIONS

TEXT INTRODUCING CURRENT OWN CODE PROCESSING POINT

SPECIFY OWN CODE AT CURRENT POINT

N

Y

ALL PROCESSING POINTS OFFERED

CHANGE TO NEXT PROCESSING POINT

Y

N

DIALOGUE ABOUT USER'S COMMENTS ON THE SYSTEM

N

GENERATE CONSTANTS MACRO

FINISH
of Appendix IX and the outline processing of the STAGE18 macro is illustrated in Figure 7.35.

7.26.1 STAGE18 macro notes

1. Copies of the condition specifying and validation routines are included in the STAGE18 macro (Appendix VII Sections 15.1 and 15.2).

2. The identification and validation of own code statements is described in Section 27 of Appendix VII together with details of the generation of the equivalent COBOL statements.

3. The general form of the OWN name prefix macros generated in this stage is shown in Section 26 of Appendix VII.

4. Section 28 of Appendix VII gives details of the variables whose values are recorded in the CONSTANTS macro and also the general form of this macro.

5. If the totalling facility was invoked in Stages 4 or 5, i.e. if global variable GTOT is equal to 1, the user is instructed how to access the subtotals and totals in his own code statements.
Figure 7.36 Outline processing for the HELP macro
The HELP macro carries out a dialogue with the user in which
the 'help' facilities are described and the selected option
implemented. The detection of a plea for help and the facilities
available are described in Section 6.2.3. The HELP macro linkages
are illustrated in Figure 7.2 and the outline processing is shown
in Figure 7.36.

One and sometimes two text subfiles are used as input to the
HELP macro. The DIALHELP subfile contains the text describing the
help facilities. The other text subfile MODELn, where n is the
current stage number contained in global variable $CKPT, is used
only if the option to view the dialogue for the corresponding
stage of the model problem is selected.

The WORK subfile is used as a 'sink' in which to empty the
PG/2 macro processor output stack of any statements generated in
the stage from which the plea for help was made.

The processing details for the HELP macro are given in
Section 29 of Appendix VII and a listing of the partially developed
macro is included in the separate folder (Item 28).
Figure 7.37 Outline structure of the PG/2 macro processor input file for the generation of a complete COBOL report program

START

System macros to initialise the PG/2 macro processor

System macros to load constants and other 'grown' macros

Call constants macro to restore string and global variables

Define and call first COBOL generating macro

System macros to save generated COBOL statements in COBOLPROG

System macros to reset PG/2 macro processor

System macros to load constants and more 'grown' macros

Call constants macro to restore string and global variables

Define and call next COBOL generating macro

System macros to append generated COBOL statements to COBOLPROG

--- etc ---

FINISH
In the process of generating a complete COBOL report program, the program is built up by synthesising a very large number of fragments of code (rather like conventional compiler code generation). This is in contrast with the process of generating a FORTRAN program [5] which consisted of 'filling out a framework'.

During the problem specifying dialogue the user supplied all the information necessary to generate the complete COBOL program. This information was recorded in the common data subfiles and the macro definitions which were 'grown' during the various stages of the dialogue. The complete COBOL program can be generated using the PG/2 processor and a prepared input file containing a series of system and COBOL generating macros. Figure 7.37 illustrates...
the general form of the PG/2 macro processor input file which causes the complete COBOL program to be generated and filed in the COBOLPROG subfile.

The generation of the complete COBOL program is carried out by a series of COBOL generating macros for the following reasons:

1. A single macro would be too large.
2. The PG/2 stacks are not large enough for all the 'grown' macros to be loaded at the same time.
3. The PG/2 output stack is not large enough to contain the complete COBOL program.

The structure of the COBOL program is partly determined by generator macros and partly by the 'grown' macros. The program listing in Appendix X is annotated to show which statements come from 'grown' macros and which come from the generator macros.

The system macros control the loading of the 'grown' macros, the bodies of which, with the exception of the CONSTANTS macro, contain fragments of COBOL code. The COBOL generating macros make nested calls to the 'grown' macros so that the COBOL code they contain is regenerated in the correct position in the program. In the COBOL generating macros the calls to the 'grown' macros are interspersed between macro-time statements which generate other fragments of code. The information from which these fragments are generated is obtained from the common data subfiles and the values of the string and global variables as set by the CONSTANTS macro.

During the COBOL generating macro which carries out the Data division preliminaries (Appendix VIII Section 3) the common data subfile PICTUREDICT is written and two more macros, TUPLE and SAVETOTL, are 'grown'. They are used to record information and COBOL statements which are required by subsequent COBOL generating macros.
Details of the generation of a COBOL report program, for the general case where the user's data is on more than one input file and sequenced on more than one key are given in Appendix VIII. All the data file sequence keys are assumed to be in ascending order and there is no provision for the elimination of duplicate detail lines.

A useful extension would be to amend the generator macros so that they could take advantage of special circumstances, e.g. where only one data file is used or when all the data files are sequenced on one and the same key. The generated program could then be structured to take advantage of the special conditions and so yield simpler COBOL code. Areas where the special conditions prove advantageous include the totalling facility, the matching of input record keys and the detection of sequence key breaks.

A listing of the COBOL program which would be generated for the solution of the model problem described in Section 5.9 is included in Appendix X.

7.29 SUMMARY AND APPRAISAL

The material presented in this chapter and the contents of Appendices V to X demonstrates the feasibility of generating valid COBOL report programs from information gathered in a dialogue with a casual user who is presented with a relational view of his data.

The use of the catalogue enables data access to be controlled and allows the relational view to be presented without burdening the casual user with details of the physical organisation of his data files. Although only basic facilities were developed in the Librarian macros to enable the data base administrator to maintain the catalogue of file descriptions, there is potential for these to be enhanced so that greater use
can be made of the catalogue information.

By carrying out problem specification in foreground mode and relegating the generation of the complete program to background mode, the COBOL generating system contributes to the effective use of computer resources. The segmentation of the problem specifying dialogue into separate stages, each dealing with a different aspect of the report specification and consisting of one or more macros, contributes to the ease with which the system can be maintained and enhanced. Additional features can be implemented by inserting more stages in the dialogue and/or more macros in the chain.

The processing described for the COBOL generating macros and the manually fabricated program show that syntactically correct COBOL can be generated which will produce the report requested in the problem specifying dialogue.

The COBOL report program generating system is large and relies heavily on the use of the PG/2 macro processor subfiles for the storage of macros, instructional text, common data, the generated program and the catalogue of file descriptions. The extensive validation of the user's responses and the relatively low level language of the macro processor are factors which contribute to the volume of coding required to implement the macros. The relatively poor diagnostic facilities of the macro processor (4.8) tend to increase the time required to develop the large macros. The inefficient implementation of the macro-time statements for accessing the subfile records (4.8.2) makes the developed macros 'peripheral limited', while the lack of the stack emptying facility (4.8.1) precludes the complete development of the system beyond the fourth stage of the problem specifying dialogue.

The quality of the generated code compares favourably with
hand coded programs although there is still some scope for improvement. Many of the generated COBOL statements occupy two lines of coding although, if hand coded, they require only one. This is because in the generated statements space is always allowed for data names to consist of 15 characters, the maximum permitted. The speed and efficiency of the COBOL compiler more than compensates for the additional macro-time statements which would be required to improve the elegance of the generated code in this respect.

It would be possible to amend the generation strategy to optimise certain aspects of the generated program. Some examples of where this can be done are given below.

At present the statements for setting up the run time and date items ZTIME and ZDATE are always generated even if these items do not appear in the printed report. Similarly the page number item Z-PAGE-COUNT is generated and incremented even when the report pages are unnumbered. These program inefficiencies can be eliminated by making the generation of the COBOL statements conditional on global variable indicators set during the problem specifying dialogue.

The statements which define and process the Z-IGNORE data item are also generated unconditionally. Ideally, they should be generated only if the user chose to ignore data base inconsistencies in Stage 6 of the problem specifying dialogue, i.e. if global variable #GOPT was set equal to 1.

Another part of the COBOL program which merits further study is the use of the Z-CONTROL-BREAK group. As this group duplicates the sequence key items in the Z-TUPLE group, an investigation into the advantages of making Z-CONTROL-BREAK a subgroup of the Z-TUPLE group would be worthwhile.

The strategy adopted in the Z-FETCH-TUPLE section of the
COBOL program for matching sequence keys of non-master input files relies on the use of the RETAIN option of the CLOSE verb. The RETAIN option is not available for use with files on cards or paper tape, but there is no check to prevent such an error being generated. Attempts to use input files on cards or paper tape can be detected in the STAGE3-1 macro of the problem specifying dialogue. Additional validatory steps can be provided when the checks for data base inconsistencies are made.

The effectiveness of the COBOL generating system can be fully appraised only when a complete operational system is available.
CHAPTER 8

APPRAISAL AND CONCLUSIONS

8.1 INTRODUCTION

In this chapter the various aspects of dialogue based COBOL report program generation are appraised. Areas which can be improved or merit further study are identified and conclusions are drawn. The material is presented under the following headings:

1. Relational view of the data.
2. Security procedures.
3. User's approach to the system.
4. Instructional text.
5. Magnitude of the COBOL generating system.
6. The generated COBOL program.
7. Towards more effective computer usage.
8. Aspects for further study.

8.2 RELATIONAL VIEW OF THE DATA

The system described in Chapters 6 and 7 has the advantage of being able to present the user with a relational view of his data at the user interface. No knowledge or understanding of the physical organisation of the data is required of the user, he has only to visualize it in tabular form.

The COBOL generating system makes use of flat data files which already exist and were not specifically designed for use in a relational data base. Each file is considered to be a relation and each field within the file to be a domain. Several relations may be joined to provide the data base submodel required for the
solution of the user's problem (5.3). Only sequence key fields may appear in more than one relation. If more than one data file is used to solve a problem, the generating system ensures that the names of all non-key fields will be unique when they are listed prior to problem specification (6.4.2).

8.3 SECURITY PROCEDURES

User access to the data is controlled. The data in the database is protected by means of relation names each with a set of multi-level passwords (6.3.1). Thus each user need only be aware of a submodel of the database and different users may have different views of it depending on the relation passwords allocated to them by the database administrator. The security system protects the casual user from being overwhelmed by a large volume of superfluous data and at the same time restricts access to sensitive data.

As the files used by the COBOL report program generating system are created and maintained by other systems, it is assumed that they ensure the integrity of the data and provide adequate 'back up' should a file become corrupted. The integrity of the catalogue of file descriptions is the responsibility of the database administrator. At present the generating system relies on the standard file dumping procedures of the computer installation for the recovery of the catalogue and macro definition subfiles in the event of a failure. This approach is inadequate for an operational system, especially if its facilities are extended to permit the updating of data files. The design and implementation of additional features to maintain the integrity of the system and the database should form part of any future extensions of
Before engaging in the problem specifying dialogue the user must have identified his problem and know what data is required to solve it. He should also have an outline idea of the format in which he wishes the report containing the answers to his problem to be presented.

A previous user of the system will already have been supplied with the relation names and passwords which define the bounds of his data base. A first time user will need to consult the data base administrator in order to find out these details.

Although the problem specifying dialogue attempts to be self-teaching there is a great deal for the first time user to assimilate. In order to demonstrate the potential of the COBOL report program generating system and to complement the instructional aspects of the dialogue, an introductory booklet should be available to each first time user. The booklet should be based on the material in Section 5.9 of this thesis, and should contain a description of the model problem, details of the data base and its temporary extensions, an outline of the required report format, a listing of the problem specifying dialogue and samples of report output produced by the generated COBOL program.

Another aspect of 'getting to know and use' the system is the accuracy with which errors in the user's responses are detected and the ease with which they can be corrected. As all the user's responses are validated a considerable portion of the macro-time statements in each problem specifying macro are devoted to validation processing. When an error is detected a meaningful error message is output and the user is prompted to enter an amended response. The 'help' facility has an important role to
play in this respect. In circumstances where the user's terminal does not produce 'hard copy' it may be advantageous to 'journal' all the conversation so that it can be available for study.

8.5 INSTRUCTIONAL TEXT

Whilst the text illustrated in Appendix IX demonstrates that it is feasible to devise instructional text which prompts the casual user to specify his request for a report, the efficacy of the instructional text requires further assessment.

In order to make a qualitative assessment of the instructional text, the dialogues with numbers of users, whose requests for a report cover a wide spectrum of applications, should be analysed. This underlines the need for an operational system as a basis for further research.

Users' comments:

The comments entered by users at the conclusion of the problem specifying dialogue would provide one source of assessment data.

Analysis of errors:

More revealing, perhaps, would be the user responses to dialogue prompts which, when validated, cause error messages to be output. These would serve to highlight the less effective stages of the instructional text. The dialogue stages where the user has to enter other than single character responses are expected to yield the most information, e.g. condition specification, editing, line formats and own code statements.

Calls for 'help':

Another possible source of material for assessing the
effectiveness of the instructional text is the HELP macro. This could be enhanced to include a facility for logging details of the dialogue stage from which the macro was called and the help option selected.

System familiarity:

As a user becomes more familiar with the instructional text it is likely that he will not wish to waste valuable time waiting for all of it to be output on his terminal each time he specifies a request for a report. This is especially true if the user is working at a teletype terminal, but it can still prove irritating with a faster peripheral.

Macro processor facilities for suppressing all or selected parts of the instructional text already exist in the form of the %QUIET and %TALK system commands and in the WRITEOFF and WRITEON macro-time statements (Appendix I). A description of how to use these system commands can readily be incorporated in the instructions to the user in Stage 1 of the problem specifying dialogue. The macros for the various problem specifying stages are easily amended to incorporate WRITEON macro-time statements. These are used to override the %QUIET system command and force the dialogue back into 'talk' mode so that requests for information and error messages appear on the user's terminal.

8.6 MAGNITUDE OF THE COBOL GENERATING SYSTEM

The macros which create and maintain the CATALOGUE subfile containing the data base file descriptions have been developed and their listings are available in the separate folder. Excluding comments, approximately 800 macro-time statements were required to code these macros.

Coding details for the macros of the first eight stages of
the problem specifying dialogue are also available in the separate folder, but the development of macros beyond STAGE3 is incomplete. Some 2100 macro-time statements, excluding comments, were used to code these macros. It is estimated that the coding needed to implement the macros for all stages of the problem specifying dialogue would extend to between 5000 and 5500 macro-time statements.

The macros which generate the complete COBOL program from the common data subfiles and the 'grown' macros would, it is estimated, require a further 900 to 1000 macro-time statements.

The experience gained while developing macros for the system showed that, although the macro processor language is very suitable for generating COBOL statements, it is at rather too low a level for response validation or for accessing the contents of fields within subfile records. For example, higher level language features similar to the Free format READ of certain FORTRAN dialects or the COBOL CLASS Condition test for identifying numeric and alphabetic strings would facilitate the coding of macros.

As described in Section 7.3 the COBOL report program generating system relies heavily on the PG/2 macro processor filing system for the storage of a large number of macro definitions, text and common data subfiles. This extensive use of disc storage tends to make the generating system 'peripheral limited' (7.29). If the macro processor language supported array type variables, the use of subscripts would enable the number of common data files to be reduced.

The COBOL generating system does, however, demonstrate that good COBOL code for solving the user's problem can be generated from the information entered by the user during a, albeit lengthy, computer dominated dialogue.
8.7 THE GENERATED COBOL PROGRAM

Although the improvements identified during the Filetab to COBOL translation study (3.7) were incorporated in the design of the generated COBOL report program, there is still some scope to make further improvements, in order to make it comparable with the best hand coded program (7.29).

It seems likely that with further work it would be possible to revise and amend the generation strategy to optimise the generated COBOL report program. The generation process could also be generalised to cope with other problems.

8.8 TOWARDS MORE EFFECTIVE COMPUTER USAGE

Assuming that all the implementation problems associated with the PG/2 macro processor (4.8) can be resolved, it is estimated that, using the COBOL report program generating system, a casual user would require a terminal session of between 3 and 4 hours to specify the model problem described in Section 5.9. The estimate is based on the length of the problem specifying dialogue, which is assumed to be conducted on a teletype capable of printing at 10 characters per second, with due allowance for 'thinking time'. With a faster terminal, the time required to output the instructional text would be significantly reduced, say by a factor of 3 or 4.

In order to assess the contribution that the COBOL generating system could make towards more effective computer usage, comparisons of the man hours required to produce a similar report using other software systems should be made. Ideally the comparisons should include the use of other report generators such as Filetab as well as the development of hand coded COBOL programs.
Investigations carried out by Chrysler [21] into program development times suggest that these depend on the experience of the programmer and certain program characteristics, particularly those associated with the number and complexity of the files used. These findings are corroborated by the estimating procedures currently in use by a Software house [22]. These estimating procedures take into account the difficulty of the problem, the estimated size of the COBOL Procedure division (excluding paragraph and section names) and the number of files used by the program. This information is used in a set of confidential formulae to predict the number of hours an average programmer would take to flowchart, code, test and document a program.

The model problem (5.9) uses three input files and one output file and was assessed to be of average difficulty. In order to take account of the potential improvements noted in Section 7.29, it was assumed that the hand coded Procedure division would be only 95% of the size of the generated one (Appendix X). This data used in the above mentioned formulae yielded a figure of 58 programmer hours, excluding documentation time, as the estimated development time for a COBOL program to solve the model problem.

Not only does the COBOL report generating system show considerable potential for a saving of man hours, which could be of the order of 95%, it allows the same task to be achieved by a person with much less technical skill and training. A skilled user with an up to date video terminal would be able to make even more effective use of the system, for he would be able to dispense with much of the instructional text and concentrate on the problem specifying dialogue. With effective response validation, routine testing and rewriting to remove ‘oversights’ would be virtually eliminated.
The use of the COBOL generating system would influence program maintenance. It would be possible to extend the system so that important parts of the dialogue are filed. Facilities could then be provided for the user to vary the dialogue as it was reloaded and then the amended program could be generated.

Another application of the COBOL generating system would be to adapt it so that it generated programs in the COBOL dialect used in a micro computer. Thus inexperienced users could generate their COBOL programs on the mainframe computer and transfer the source code to the micro computer for compilation and execution.

When a fully operational version of the COBOL report generating system has been developed, there is scope for gathering data to investigate savings in the use of computer resources. The terminal connect time and CPU or 'mill' time usage required to generate a COBOL report program can be compared with those required to develop a hand coded COBOL program to produce the same report. The production run time statistics for the generated and hand coded programs can also be compared.

8.9 ASPECTS FOR FURTHER STUDY

The aspects for further study fall under six main headings:

1. Those related to the PG/2 macro processor.

2. The extension of facilities offered during the report requesting stages.

3. The extension of the technique to other information processing problems.

4. The benefit to the skilled user.

5. Consideration of alternative host computers and languages.

6. Other applications.
8.9.1 PG/2 macro processor

In addition to the provision of the stack emptying facility and improving access to subfile records as discussed in Section 4.8, some other aspects of the PG/2 macro processor merit further study.

First there is a need to consider the provision of better diagnostic facilities during the development of macro definitions. At present errors in macros are indicated by the output of messages in the form 'ERROR NO. nn' (Appendix I Section 9). As there is no indication as to which statement the message refers, it is often difficult to identify, especially in large macro definitions.

If an error occurs during the execution of a macro it would also be advantageous for the user to be able to request the output of local, global and string variables. A trace-back facility would also help to determine how far the macro evaluation had proceeded.

The facility to 'grow' macro definitions is another aspect of the PG/2 macro processor which merits further study. Only relatively simple macros were 'grown' during the investigations into the generation of COBOL programs. There is, however, scope for examining the problems of 'growing' more complicated macro definitions, especially those which contain arguments, CHAIN statements or nested calls to other macros. If such a study established the feasibility of 'growing' macros containing CHAIN statements, there is potential for improving the efficiency of the the third phase of the system which generates the complete COBOL program. The loading of 'grown' macros with empty bodies could be greatly reduced.

As the PG/2 macro processor is machine dependent the use of
other languages for COBOL generation is briefly considered in Section 8.9.5.

8.9.2 Extension of facilities offered during the report requesting dialogue

In addition to the facilities already proposed but not yet implemented, e.g. the wide report page and the default formatting of report lines, the following paragraphs highlight some other possible extensions.

The 'help' facility currently offers only 4 options, but the structure of the HELP macro is such that other options can readily be added. Some of those which might merit inclusion are considered below:

1. Access to a glossary of technical terms, e.g. relation, domain, etc.

2. Access to the contents of the LABELTABLE subfile in order to be reminded of the label character assigned to a particular domain for the purposes of format specification.

3. Access to the DOMAININDEX subfile so that the user can be reminded of the size and type of data in a given domain or temporary item.

4. To provide a LIST facility whereby the user may (possibly selectively) inspect the CATALOGUE subfile definitions of all files and fields to which he has password access.

A facility to record the details of the user's complete problem specification could be provided. This would enable minor changes to be made at a later date without the need to repeat all the stages of the dialogue. If, as discussed later (8.9.3 and 8.9.4), the program generating system were extended and generalised for use by programmers, this facility would become more important. It would provide a means of program documentation
and a basis for program maintenance and future development.

Another aspect which merits development is the extension of the methods of file organisation and data types which can be processed by the generated COBOL program. At present there is provision for only sequentially organised data files whose records contain numeric and alphanumeric character fields with DISPLAY usage. Already the catalogue of file descriptions in the CATALOGUE subfile anticipates that direct access and indexed sequential files will be included. The implementation of this and the inclusion of other data usages, e.g. COMPUTATIONAL, for the fields of input records would extend the usefulness of the COBOL generating system. There is also potential for some relaxation in the key matching criteria imposed when the user wishes to use more than one data file (relation) to solve his problem (6.4.2). These enhancements would extend the data base available to the user.

At present the problem specifying dialogue does not warn the user when a problem domain or temporary item (6.6) is neither referenced by own code statements nor included in the report. Also there is no provision for detecting 'size errors' which result from the totalling facility or arithmetic own code statements. The provision of additional validation in respect of these would make a small contribution towards ensuring that a semantically correct COBOL program was generated. An operational system which can be used, and modified in the light of experience, is needed to see how far one can go towards obtaining semantically correct programs. In the end only the user can say whether the COBOL program did what he intended!

There may be times when the report requested by the user contains only items from non-key fields. As not all fields can be treated as domains (6.3.2) the report may contain duplicate
detail lines. The option for duplicates to be omitted from the report could be offered in Stage 10 of the problem specifying dialogue. The provision of this facility would in certain circumstances require the generated program to contain SORT statements and include additional file descriptions for sorting work files.

There is also the possibility of adding new powerful macros to speed up the request processing. These macros would be defined either by the database administrator or by a skilled user.

8.9.3 The extension of the technique to other information processing problems

Once the efficacy of the self-tutorial approach has been established, the range of information processing problems which can be solved by the generation of a COBOL program can be extended to include, for example, file updating. Such an extension would require changes to the problem specifying dialogue and the provision of more own code facilities. The structure of the generated COBOL program would have to be reorganised to process additional input and output files.

8.9.4 The benefit to the skilled user

Although designed primarily for the casual user, the man-hour estimates quoted in Section 8.8 show that the COBOL generating system is potentially of great benefit to the skilled user.

With a skilled user a shorter problem specification dialogue would be needed as the instructional text could be dispensed with. Dialogue changes could also be made to take advantage of the programmer's ability to use a more sophisticated terminal device, e.g. light pen, data pad, etc. The availability of such devices
would prove particularly beneficial for the design of report page layouts.

8.9.5 Consideration of alternative host computers and languages

The PG/2 macro processor is written in PLAN, the assembly language of the ICL 1900 computer series. The life of this range of computers is limited as it is being superseded by the ICL 2900 series of computers. This situation raises a number of points which merit further study.

Should effort be devoted to converting and restructuring the PG/2 macro processor for use on the ICL 2900 series? This would facilitate further development and evaluation of the COBOL generating macros described in Chapter 7 together with the improvements and additional facilities outlined in this chapter. Such an approach could be considered only if it was worthwhile running the PG/2 macro processor using the 1900-emulation package. This would be a short term solution because of the defects in the macro processor software which cannot be easily or cheaply remedied.

Alternatively, would it be beneficial to implement the self-teaching COBOL generating system in a more universal host language? This would certainly make the system more portable for implementation on other makes of computer. In general the high level computer languages have facilities for program overlay, direct and indexed sequential file access, array storage and good error diagnostic facilities. Facilities for the manipulation of variable length strings, data validation and statement generation are not so generally available. Often, however, these facilities can be provided by means of subroutines which may have to be coded in assembly language.

Consideration needs to be given to ways of adding to the
high level language facilities and discovering how far they could be coded directly in the high level language. As there are currently moves to enhance the COBOL language facilities, this is one of the obvious high level languages to consider for the development of a COBOL generating system. A working paper prepared by the British Computer Society COBOL Specialist Group [23] contains a justification and description of proposed language enhancement features for COBOL which have already been proved by a pilot implementation. Although in June 1980 the CODASYL COBOL Committee rejected these proposals by a small majority, work continues on refining and improving them.

The experience gained during the practical work of this project provides the basis for an initial attempt at defining the range of facilities required for the generation of COBOL programs (Appendix VI).

8.9.6 Other applications

The number of micro-computers will soon greatly exceed the number of programmers available, so there is scope for developing ways of making them easier to use by non-specialists. As there is a growing tendancy for a dialect of COBOL to be offered for use on such computers, the provision of facilities for generating COBOL programs would contribute to their effective use. As mentioned in Section 8.8, the source code for programs in a micro-computer COBOL dialect could be generated on a main frame computer and transferred to the small computer for compilation and execution. There is also scope for investigating to what extent program generation facilities could be provided using the micro-computer itself.
8.10 CONCLUDING SUMMARY

The project set out to see if, by the generation of COBOL programs, access to the computer by casual users could be made easier. What has been done and learned as the result of this project can be summarises as follows:

1. A dialogue can be created which can be used to generate COBOL programs.

2. Validating the dialogue responses and generating COBOL is a major undertaking leading to a system comparable in size and complexity to that of a compiler.

3. The PG/2 macro processor is not an ideal tool but it has helped to discover the characteristics of an appropriate tool.

4. Program generation appears efficient but may be of even more use to the professional programmer than to the casual user.

5. An operational system is needed to validate the potential of this approach.

6. The project can be easily and naturally extended in many ways, thus providing a continuing opportunity to contribute towards easier and more effective computer usage.
### APPENDIX I

**PG/1 MACRO PROCESSOR**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. MACRO PROCESSOR STRUCTURE</td>
<td>2</td>
</tr>
<tr>
<td>2. STATEMENT LAYOUT</td>
<td>2</td>
</tr>
<tr>
<td>3. MACRO DEFINITION FORMAT</td>
<td>3</td>
</tr>
<tr>
<td>4. MACRO CALLS</td>
<td>6</td>
</tr>
<tr>
<td>5. EXPANSION-TIME STATEMENTS</td>
<td>8</td>
</tr>
<tr>
<td>6. SYSTEM MACROS</td>
<td>11</td>
</tr>
<tr>
<td>7. MACRO-TIME STATEMENTS</td>
<td>15</td>
</tr>
<tr>
<td>8. RESERVED GLOBAL VARIABLES</td>
<td>22</td>
</tr>
<tr>
<td>9. ERROR MESSAGES</td>
<td>24</td>
</tr>
</tbody>
</table>
APPENDIX I

PG/1 MACRO PROCESSOR

The PG/1 macro processor may also be considered as a general purpose string processor because its macro-time facilities allow the extraction, identification, manipulation and concatenation of string variables.

The material provided in this appendix has been extracted from a report of a earlier investigation [5]. It gives a description of the PG/1 macro processor used for the preliminary investigation into the feasibility of generating COBOL programs. The material is subdivided into the following sections:

1. Macro processor structure
2. Statement layout
3. Macro definition format
4. Macro calls
5. Expansion-time statements
6. System macros
7. Macro-time statements
8. Reserved global variables
9. Error messages
1. MACRO PROCESSOR STRUCTURE

The PG/1 macro processor has two buffers each capable of storing up to 72 characters, one buffer for input and the other for output. There are also storage areas used for label, symbol and pointer tables and stacks available for macro expansion statements, names or definitions and saved blocks of output text.

A disc-based filing system is available in the form of a collection of subfiles each of which may be used to contain macro definitions, partially developed macros or any other character data. In addition to the macro processor PG/1, two other programs are used to maintain the disc filing system. The filing system is initialised using the first of these programs and, once initialised, may be maintained by the second program. This latter program enables the user to allocate space for the creation of new subfiles and also has facilities for the input and output of character data, and for editing copying and deleting subfiles.

2. STATEMENT LAYOUT

The format of source input to the macro processor follows standard FORTRAN convention with labels in positions 1 to 5 and statements in positions 7 to 72. (The use of column 6 as a continuation record was suspended as mentioned in Section 2.2.4). The FORTRAN convention arises from the fact that initially the PG/1 system was used to generate FORTRAN programs. Where labels are generated by the system these are FORTRAN type integer labels, but apart from these considerations the PG/1 system is not biased towards any particular source language.

The original MP/1 system [6] was designed to be used both as a macro processor and a pre-processor and consequently the format of macro names was made less restrictive than is the case for most macro systems.
3. MACRO DEFINITION FORMAT

The macro definition specifies both the macro name and the associated replacement body or expansion string. A definition is initiated by the pseudo macro %DEF. The pseudo macros and all the system macros start in position 1 of an input record. The macro name follows and is taken to terminate at the end of the statement record. The subsequent statements up to the pseudo macro %END are taken to constitute the macro body. Definitions are written using the ICL 64 character set although some special characters are excluded.

The macro name is a character string interspersed with formal parameters where desired. Two or more formal parameters must not appear consecutively in a name as the intervening sub-strings between parameters of the name are used as argument terminators when matching macro calls. However, a single formal parameter may be defined to represent a variable number of actual parameters as described below. It is also useful, especially in the case of variable parameter lists, to be able to specify that the parameter, or each parameter of a list, should be balanced with respect of left and right parentheses. This feature is provided together with a feature equivalent to the keyword parameter of the System/360 Macro Language. In the latter, a formal parameter may have its value set in the definition and if the corresponding actual parameter in the call is omitted then the value is used as if it were the actual parameter, otherwise the preset value is overwritten by the value obtained by matching.

Using a notation based on Bakus Normal Form [8] the name definition takes the general form defined below:
\(<\text{macro name}\> ::= \{\text{arg}\}\text{<delimiter>}|\text{delimiter}\text{<macro name}>|\text{macro name}\text{<arg}>|\text{delimiter}>|\text{arg}>$

\text{<arg> ::= @<inset>} \text{3}$

\text{<inset> ::= <null> | <null>} | \text{<sep> | <sep> | <inset> = <preset list>}$

\text{<null> ::=}<\text{inset}>$

\text{<sep> ::= }^3|\text{<char>}$

\text{<preset list> ::= ' <preset argument> ' | ' <preset argument> ',} \text{<preset list>}$

\text{<preset argument> ::= }^3|\text{<char>}$

\text{<delimiter> ::= }^3|\text{<char>}$

Where \text{<char>} is a member of the character set defined above.

From these definitions it follows that the formal parameter \text{<arg>} is a sequence of one or more characters enclosed by @ symbols. When this takes the form @ @ then the parameter is a single element which need not be balanced. When the formal parameter is of the form @ <sep> @ then this denotes that the macro call will contain a list of one or more elements corresponding to this one parameter. The one, two or three characters denoted by \text{<sep>} will serve to separate the individual elements of the actual parameter list. In either of these cases if a left parenthesis immediately succeeds the first @ symbol and a right parenthesis immediately precedes the second @ symbol then a check is made to ensure that the actual parameter and each element of the actual parameter, if appropriate, is balanced with respect to the symbols ( and ). Also a parameter or parameter list may be preset as indicated above, the preset value being a list of one or more elements enclosed in quotes.
and separated by commas.

The expansion string consists of optionally labelled statements which may be source language, macro calls or expansion-time statements. Macro calls internal to a definition are enclosed in square brackets and may appear either within a source language statement, another macro call or as a separate statement. Expansion-time statements are described below.

When a macro call is expanded labels inside the definition are replaced by unique five digit labels. Labels outside columns 1-5 of source language statements must be prefixed by a character. Formal parameters may be used in the definition to show points where the corresponding actual parameters of the call are to be inserted. These parameters are similar to formal parameters in the name and take the following form:

\[
\langle \text{exarg}\rangle::= \@\langle \text{int}\rangle\langle \text{inset}'\rangle\@
\]

\[
\langle \text{inset}'\rangle::= :\langle \text{int}\rangle | :\langle \text{expansion variable}\rangle | \langle \text{sep}'\rangle
\]

\[
\langle \text{null}\rangle
\]

Where the first \(\langle \text{int}\rangle\) is an integer establishing a correspondence between the formal parameter of the expansion string and the formal parameter of the name, 1 referring to the first (left most) parameter of the name, 2 to the second and so on. Where the formal parameter is a simple one then this integer is followed by a second \(\@\). In the case where the formal parameter refers to a parameter list then the integer may be followed by a colon followed by either an integer or an expansion-time variable containing an integer. This second integer quantity will then specify the particular element of the list to be output, e.g.

\(\@2:3\@\)

would refer to the third element of the second parameter.

Alternatively the parameter list can be output as a list in which
case it takes the form
\@[int\]<sep>@

where <sep> is the character string to be used to separate the output elements. This latter string may be of any length but should not commence with either a colon or an integer.

Two examples of a macro definition format are illustrated in Section 2.2.5. The first example shows a simple macro definition without arguments while the second example shows a macro which requires one argument.

4. MACRO CALLS

A macro call will be recognised as such by a special warning character in the first position. This will be a % character unless otherwise specified by the programmer as shown below. The call may be labelled. The macro call string will consist of the delimiters of the macro name to which it refers written in the order in which they appear in the definition of the macro name. Optionally parameters may be written between these delimiters. The parameters may be simple character strings or they may contain macro calls or literals. Such nested calls are parenthesised by square brackets and nesting to an arbitrary depth is permitted.

When a macro call has been successfully matched by the system against a defined macro name then the evaluation of the associated expansion string takes place. If the macro call was labelled then this label is inserted unchanged before the first source language statement of the expansion string. In the case where this statement is itself labelled a CONTINUE statement is generated with the macro call label. Evaluation then continues with the insertion of actual parameters in place of the formal parameters. Where an actual parameter has been omitted then the preset value, if it exists, is inserted otherwise the statement in
which that parameter appears is omitted. Evaluation continues either until the end of the expansion string is reached or until processing is halted by an expansion-time statement. When the original macro call has been completely evaluated processing continues at the first internal macro call, if it exists, and continues until all internal calls have been evaluated. When the macro call has been fully evaluated the generated string is transferred to the output stack.

Macro calls are matched in sequence against defined names starting with the most recently defined macro name and thus it is necessary in most applications to carefully order macro definitions.
5. EXPANSION-TIME STATEMENTS

Expansion time statements may be used to generate object code which is dependent on the parameters of the macro cell. Expansion variables may be used as indexes, as criteria for conditional statements or as generators of internal symbol strings. They take the form:

\[
\text{expansion variable ::= \# I \{<\text{char}>\} |
\# G \{<\text{char}>\}
\]

where \(<\text{char}>\) is an alphabetic character. Expansion variables may be local to a particular macro definition or they may be global, the two cases being distinguished by the first letter being L or G respectively. Variables may be assigned either integer values or string values of up to 8 characters. Expansion variables may appear in normal source language statements in the definition in which case during evaluation they are replaced by the character representation of their value at that time.

Expansion time statements are introduced by a % character in position 1 of the statement and may have FORTRAN type labels for reference by other expansion time statements. The assignment and control statements take the form below:

\[
\langle\text{assignment}\rangle ::= \langle\text{expansion variable}\rangle
= \langle\text{expr}\rangle
\]

\[
\langle\text{expr}\rangle ::= \langle\text{integer expr}\rangle
\]

\[
\langle\text{integer expr}\rangle ::= \langle\text{int}\rangle | \langle\text{int}\rangle \langle\text{op}\rangle \langle\text{expr}\rangle
\]

\[
\langle\text{string expr}\rangle ::= \langle\text{string variable}\rangle
\]

where \(<\text{int}>\) is either a literal integer or an expansion variable holding an integer value and \(<\text{string variable}>\) is an expansion-
time variable holding a string variable. Where the \(<\text{string expr}>\)

is a parameter reference, \(<\text{exarg}>\), then if the parameter is greater

than 8 characters in length the left hand variable will be set
equal to the first 8 characters.

\[ \langle \text{op} \rangle ::= + | - \]

Expressions may include simple arithemtic operations (addition

and subtraction) between integers and variables holding integers.

\[ \langle \text{control statement} \rangle ::= \text{GOTO}\langle \text{label} \rangle | \text{EXIT} | \]

\[ \text{IF}\langle \text{rel} \rangle, \text{GOTO}\langle \text{label} \rangle | \]

\[ \text{IF}\langle \text{rel} \rangle, \text{EXIT} | \]

\[ \text{CONTINUE} \]

\[ \langle \text{rel} \rangle ::= \langle \text{IR} \rangle.\langle \text{IOPR} \rangle.\langle \text{IR} \rangle | \]

\[ \langle \text{SR} \rangle.\langle \text{SOPR} \rangle.\langle \text{SR} \rangle \]

\[ \langle \text{IR} \rangle ::= \langle \text{int} \rangle \]

\[ \langle \text{IOPR} \rangle ::= \text{EQ}|\text{NE}|\text{LE}|\text{LT}|\text{GE}|\text{GT} \]

these six predicates being equal, not equal, less than or equal,

less than, greater than or equal and greater than.

\[ \langle \text{SR} \rangle ::= \langle \text{string expr} \rangle \]

\[ \langle \text{SOPR} \rangle ::= \text{EQ}|\text{NE}|\text{IN}|\text{AT} \]

The operation IN and AT in this case test if the first string is

a substring of the second and if the first string is the first

substring of the second, respectively.

The GOTO statement causes an unconditional transfer of

control to the labelled expansion statement. EXIT causes 

evaluation of the current expansion string to stop and has the same

effect as END. The IF statement causes transfer of control if

the specified relation is true and the CONTINUE has no effect,

being used only as a label reference for other assembly time

statements.
The statement

\[ \text{CALLARGS}(<\text{expansion variable}>, <\text{formal parameter}>) \]

where

\[ <\text{formal parameter}> := @<\text{int}@ \]

sets the specified expansion variable to the number of elements contained by the associated actual parameter at the time at which it is called. If the parameter has been omitted the variable is set to zero.
6. SYSTEM MACROS

The system macros are used as data control operations and there are fourteen system macros in all.

6.1 %MACRO

The first record input to the PG/1 system must be of the form

%MACRO.

Optionally up to three characters may be inserted before the full stop. The first of these defines the macro warning character, the second the formal parameter marker and the third the expansion label marker.

Thus for example

%MACRO*.

would cause the % warning character and the $ symbol to be replaced by % and * respectively. In this case the label marker # would remain unchanged. This macro outputs the date and time together with a list of the three warning characters whether or not these have been redefined. All data following this record is listed but otherwise ignored until the %BEGIN statement is encountered.

6.2 %BEGIN

This statement causes all subsequent records to be treated as input to the macro processor. Statements which are part of neither macro definitions nor calls will not be altered in any way. The entire system is initialised by this statement. (See Section 2.2.5 for an example).

6.3 %END

This statement causes the output stack to be listed. (See Section 2.2.5 for an example).
6.4. **%FINISH**

The input to the macro processor is terminated by this statement. (See Section 2.2.5 for an example).

6.5. **%FILE**

The form of this command is:

```
%FILE,p,<subfile name>
```

It causes all the records following this command until an eight character record is encountered to be stored in the subfile referenced. p=0 starts storing from the top of the subfile; p=1 starts storing from the end of the current contents of the subfile.

```
%FILE,0,INPUTDATA
record 1
record 2
record 3
record 4
```

causes four records to be stored at the top of subfile INPUTDATA.

6.6. **%LIST**

This command will cause the names of all the macro definitions currently held in the store to be listed.

6.7. **%LOAD**

The form of this command is:

```
%LOAD,<subfile name>,......,<subfile name>
```

This command loads into the PG/1 stacks the current contents of the subfiles referenced. Up to four subfile names may be given.
A subfile may hold one or more macro definitions, partly developed definitions or any other data. (See Section 2.2.5 for an example).

6.8 %PRINT

The current contents of the PG/1 output stack will be printed out by this macro. (See Section 2.2.5 for an example).

6.9 %QUIET

This command sets the translator-user dialogue into silent mode i.e. supress written messages.

6.10 %RESTART

The user may delete the translator-user dialogue carried out so far, re-initialize the expansion stack and re-set the system to accept a fresh macro call by means of this command.

6.11 %SAVE

The form of this command is:

%SAVE,p,<subfile name>

The command causes the current contents of the PG/1 output stack to be saved in the referenced subfile. Parameter p plays the same role as described with %FILE above. (See Section 2.2.5 for an example).

6.12 %START

The purpose of this macro is to load into the PG/1 system the contents of the subfile "EXPLANATIONS". This subfile holds macro definitions which, when called, provide explanatory dialogue on the PG/1 system and its facilities.
6.13 %TALK

This macro will set the translator-user dialogue into its normal mode, i.e. nullify the %QUIET command and re-issue written messages.

6.14 %DEF

The function of this macro is described in Section 3 of this appendix.


7. MACRO-TIME STATEMENTS

These statements are introduced by a '%' character in the first position of the statement and may have FORTRAN type labels for reference by other macro-time statements. These statements are listed below alphabetically.

7.1 CALLARGS

Determine the number of elements contained in a referenced call parameter. (Details in Section 5 of this appendix).

7.2 CALLCONV

Convert an integer held in character form to its binary equivalent.

\[
\text{% CALLCONV(\langle m-t. vble\rangle,\langle m-t. vble\rangle)}
\]

where \(\langle m-t. vble\rangle\) may be either a local or global macro-time variable and they may be combined at will.

\[
\text{e.g. \quad \% CALLCONV(#LVAL,#GVAL)}
\]

7.3 CALLCOPY

Copy the character string, or the contents of a referenced macro-time variable, into the macro-time output buffer starting from the specified character position. If the latter is not explicitly given then use the value held in the reserved global variable OUP.

\[
\text{% CALLCOPY(n,\langle message\rangle)}
\]

where \(\langle message\rangle \equiv \langle\text{text}\rangle\mid \langle\text{macro-time variable}\rangle\)

\(\langle\text{text}\rangle \equiv \text{any combination of the ICL 64 character set and } n = 0, 1, 2, \ldots, 72.\)

The \(\langle\text{message}\rangle\) must not exceed 20 characters in length. This is a compromise figure, being typical of the maximum length of a system user message in scientific applications.
The effect of this statement is to copy the message into the macro-time output buffer, starting at the nth character position, and replacing each % character in the message by a space character.

\[
e.g. \% \text{CALLCOPY(1,THIS\%IS\%AMESSA\%G!)}
\]

7.4 \text{CALLFSTR}

Copy a fixed-length string variable from the input buffer into a referenced macro-time variable. If the values for the character position in the input buffer and the length of the string to be copied are not explicitly specified, then use the values held in the reserved global variables INP and OUP respectively.

\[
\% \text{CALLFSTR(<m-t. vble>,i,j)}
\]

which stands for CALL Fixed STRing, and where

- \(<m-t. vble>\) is a local or global macro-time variable
- \(i\) is an integer value 0, 1, 2, ..., 71
- \(j\) is an integer value 0, 1, 2, ..., 8
- \(1 < (i+j) \times 72\)

The effect of this statement is to assign to the specified macro-time variable \(j\) characters, starting from the \(i\)th character position in the input buffer.

\[
e.g. \% \text{CALLFSTR(#LVAL,1,1)}
\]

7.5 \text{CALLVSTR}

Copy a variable-length string into a referenced macro-time variable starting from a given character position until either a specified marker is encountered or 8 characters have been copied. If the value of the character position and the marker are not explicitly given, then use the values held in the reserved variables INP and MAS respectively.
which stands for CALL Variable STRing, and where

\(<m\text{-}t. \text{vble}\) is a local or global macro-time variable

\(i\) is an integer value 0, 1, 2, ..., 71

\(<\text{marker}\) is any character of the ICL 64 character set.

The effect of this statement is to assign to the specified macro-time variable all characters up to but excluding the \(<\text{marker}\) character when it is encountered.

\(\text{e.g.} \% \text{ CALLVSTR})(#LV3, 1, 5)\)

7.6 CONTINUE

This statement has no effect. It is used by macro-time statements as a label reference for other records in the macro body. (See also Appendix I Section 5.)

7.7 COPY

Copy the corresponding SAVE block. This statement is executed at output only.

\(\% \text{ COPY } n\)

where \(n\) is an integer.

This statement is used in association with the SAVE/ENDSAVE statements and causes the contents of SAVE block \(n\) to be copied into the output stack.

7.8 DEPOSIT

Copy, starting at the first available 'unfilled' character position of String variable \(S_n\), the contents of a referenced macro-time variable, the contents of another String variable, ordinary text or a combination of any of these. String variable is filled from left to right.

\(\% \text{ DEPOSIT}_n \text{<choice>}\)

where \(n = 0, 1, 2, \ldots\).
The effect of this statement is to copy all the information in, or referenced by, choice into (i) the string variable $S_n$ when $n = 1, 2, \ldots$, or into (ii) the output buffer when $n = 0$. The statement must be terminated by a '/' character, the significance of which becomes clear as $\langle \text{choice} \rangle$ is defined.

\[ \langle \text{choice} \rangle := \langle \text{macro-time variable} \rangle \langle \text{string} \rangle \langle \text{string variable} \rangle \]

\[ \langle \text{choice} \rangle \langle \text{macro-time variable} \rangle \langle \text{choice} \rangle \langle \text{string} \rangle \]

\[ \langle \text{choice} \rangle \langle \text{string variable} \rangle \]

\[ \langle \text{string} \rangle := \{ (\text{any combination of the ICL 64 character set excluding '}) \} . \]

The only limitation is that the total number of characters DEPOSITed into the String Variable referenced must not exceed 72. As with QALCOPY each '%' character in $\langle \text{string} \rangle$ is replaced by a space character.

7.9 ENDSAVE

Terminate the effect of the SAVE statement (see below).

7.10 EXIT

Stop the evaluation of the current expansion. This has the same effect as the system macro %END. (See Appendix I Section 5 for details).

7.11 GOTO

Transfer control, unconditionally, to the labelled macro-time statement. Label may be either explicitly specified or held in a referenced macro-time variable.

\begin{align*}
\text{e.g.} & \quad \% \quad \text{GOTO 123} \\
\% & \quad \text{GOTO } \# \text{GV3}
\end{align*}
7.12 IF

Transfer control if the relation specified is true. (See Appendix I Section 5 for details).

7.13 LABELOFF

Suspend processing positions 1-6 of all the following records; simply copy their contents unchanged.

7.14 LABELON

Nullify the function of an earlier LABELOFF statement.

7.15 NOTE

This statement has no effect. Processing simply continues at the next statement. This statement is used to document macro definitions.

7.16 OBEY

Transfer control, unconditionally, to a specified labelled statement, and set a second label into a RETURN word.

\% \ OBEY {label 1}, {label 2}

where {label 1} is a label on the first statement of the routine. {label 2} is used in conjunction with a RETURN statement to have the effect of an unconditional jump statement.

7.17 READ

Read a record, from the teletype or the card reader, into the macro-time input buffer.

7.18 READ \#Sn

Read the contents of the String Variable \#Sn into the macro-time input buffer. \#Sn is then empty.
7.19 RESET

Reset the referenced String variable ready for filling it up starting from its first character position.

\% \text{RESET} \#Sn

where \( n \) is the number of the String variable.

7.20 RETURN

Branch back to the statement labelled with the label \langle label 2 \rangle held in the corresponding OBEY statement.

7.21 SAVE

Suspend sending the following generated records into the expansion stack until an ENDSAVE statement is encountered. Instead, store these records in referenced SAVE-block at the end of the current expansion stack.

\% \text{SAVE} n

where \( n \) is the number of the SAVE-block.

7.22 SQUASH

Squash all the spaces out of the current contents of the input buffer. Store the character count of the resultant contents of the input buffer in reserved global variable STC.

7.23 TEST function

Test the contents of a specified macro-time variable to conform with the function specified and set the global variable indicator TST.

\% \text{TEST}\langle function\rangle \ (m-t. \ vble)

where \langle function\rangle ::= \text{DIGT} \ (\text{meaning single digit})
\quad ::= \text{LETR} \ (\text{meaning single alphabet})
\quad ::= \text{INTG} \ (\text{meaning integer value})
::= REAL (meaning real value)

The effect of any of these statements is to test that the value of the macro-time variable referenced conforms to the property as specified by (function). If the test fails reserved variable TST is set equal to 1, otherwise it remains zero.

\[ \text{e.g. } \% \text{ TESTDIGT(#LVAL)} \]

7.24 WRITE

Output the current contents of the macro-time output buffer on to the terminal, or line printer.

7.25 WRITE #$n

Store the current contents of the macro-time input buffer in the referenced String variable #$n.

7.26 WRITEOFF

This statement sets the translator dialogue into silent mode, i.e. it suppresses written messages.

7.27 WRITEON

This statement will set the translator dialogue into its normal mode, i.e. it will nullify the effect of the WRITEOFF macro-time statement or the %QUIET system command and allow written messages to be issued.
8. RESERVED GLOBAL VARIABLES

The following variables are reserved within the system. Their uses and the statements which effect each are indicated. They are listed below in alphabetical order.

CAL - (stands for CALL) holds a count of the number of calls so far processed. Its value is unique to each macro call.

CCT - (stands for Character Count) holds the number of characters found by the CALLVSTR statement before the <marker> specified for it is encountered.

0 ≤ CCT ≤ 8

DEF - (stands for Definition) holds the sequence number of the current macro definition. Its value is unique to each macro definition.

FLA - (stands for Flag) set equal to 1 or 0 depending on whether the marker specified for the CALLVSTR statement has been encountered or not.

INP - (stands for Input buffer) points at a character position within the macro-time input buffer. It is used by the CALLFSTR and CALLVSTR statements when these are not explicitly parametized. READ sets INP = 1.

1 ≤ INP ≤ 72

MAS - (stands for Mask) holds the <marker> to be used with the CALLVSTR statement when the latter is not explicitly parametized.
OUP - (stands for OUtPut) points at a character within the macro-time output buffer. It is used by the CALLCOPY and CALLFSTP statements when these are not explicitly parametrized. WRITE sets OUP = 1.

\[ 1 \leq OUP \leq 72 \]

SPC - (stands for SPaces) holds 8 'space' characters and may be used with the IF statement.

STC - (stands for STring Count) holds a count of the number of characters of the current contents of the macro-time input buffer. READ sets STC = 0.

\[ 0 \leq STC \leq 72 \]

TST - (stands for TeST) set equal to 1 or 0 depending on whether one of the TEST <function> statements returns a 'Yes' or 'No'.
9. ERROR MESSAGES

Errors are generated in the form:

ERROR NO n

where n is an integer defining the type of error which has been found. Definition errors will cause an error number to be output and the erroneous record discarded. Processing errors will cause expansion of the current macro definition to be terminated and thus no output will be generated from this macro. Only one error per one input record is flagged, but more than one error may be flagged in a definition.

1 - Initial %MACRO either omitted or incorrect
2 - Parameters in initial %MACRO in error
3 - Pointer table full
4 - Either name or definition stacks full
5 - Label table full
10 - Should not occur
11 - Macro cell cannot be matched
12 - More than three separators are used as element separators in argument
20 - Unbalanced macro-cells in definition
21 - Argument reference number in the definition is incorrect
22 - Unbalanced nested cells
23 - Incorrect argument format in definition
24 - Element specification reference is outside the possible range
25 - Text variable used as an element reference
26 - Second © character omitted in definition
30 - Attempt made to perform a numerical operation on a text variable
31 - Argument element in IF statement is outside the possible range
32 - Incompatible parameters in an IF statement
40 - Incorrect argument format in macro name
41 - Incorrect format of statement
42 - Label defined twice
43 - Incorrect variable format
44 - Symbol table full
45 - Statement does not exist
46 - String variables exhausted
47 - Character string too long (>72)
50 - String variable does not exist
51 - Surplus ENDSAVE statement
52 - SAVE-block referenced does not exist
60 - Incorrect label in OBEY statement
61 - Character position referenced within String variable or INBUFF beyond bounds
62 - Character transfer by CALLFSTR > 8
63 - Character transfer by CALLCOPY > 20
64 - OUTBUFF over-full (>72 characters)
65 - IF statement wrongly ended
70 - File referenced cannot be opened
71 - Error in SAVE or FILE modes specified
72 - Incorrect format of SAVE or FILE commands
73 - Subfile referenced cannot be opened
74 - Error in writing into subfile
75 - Incorrect format of LOAD command
76 - Incorrect format of PRINT command
APPENDIX II

FILETAB LANGUAGE FEATURES

1. CLASSIFICATION OF DATA  
2. FIELD DESIGNATION, CONTROL LEVELS AND REPORT PAGE WIDTH  
3. DIRECTIVE AND PARAMETER FORMATS

2
3
5
This appendix gives a brief summary of those Filetab language features implemented in this study and is based on information extracted from the NCC Filetab manual [7]. The material is subdivided into three sections:

1. Classification of data.

2. Field Designation, Control levels and Report Page width.

1. CLASSIFICATION OF DATA

Filetab classifies data fields into three types (CONTROL, TOTALLING and TRANSFER), and uses a single character to identify both the field and the field type (Appendix II Section 2).

TRANSFER fields are those which are transferred direct from the input file to the printed report. A TOTALLING field is a numeric field for which it is desired to accumulate totals. CONTROL fields are used to control the accumulation of totals and/or the layout of the report. Control fields are the sequence key fields of the input file and use of them permits the printing of totals and new headings when a sequence break occurs in a non-minor key field. For example, in the following records sequenced on three key fields sequence breaks occur between the records marked with an arrow.

<table>
<thead>
<tr>
<th>Major Key</th>
<th>Sub-major Key</th>
<th>Minor Key</th>
<th>Non-key data fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>10</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>10</td>
<td>05</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>10</td>
<td>06</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>20</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>0200</td>
<td>10</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>0200</td>
<td>10</td>
<td>09</td>
<td></td>
</tr>
<tr>
<td>0200</td>
<td>30</td>
<td>07</td>
<td></td>
</tr>
<tr>
<td>0200</td>
<td>30</td>
<td>08</td>
<td></td>
</tr>
</tbody>
</table>
2. FIELD DESIGNATION, CONTROL LEVELS AND REPORT PAGE WIDTH

2.1 FIELD SPECIFYING CHARACTERS

These characters are used to name a field in a record and to specify the way in which a field is to be used. They fall into three groups - transfer, control and totalling. The *INL directive parameters are used to name each data field required to produce the report by associating it with a character from the appropriate group.

2.1.1 Control fields

Up to six control fields are permitted and are represented by the characters M N O P Q R. M denotes the lowest or minor control level, while R denotes the highest or major control level. Starting with M these characters must be used consecutively for as many levels as are required. Thus if only three control fields are required only M, N and O are used.

2.1.2 Control levels

In addition to the six control fields defined by the above character group, two further levels, L and F, are defined. L denotes list level control, the lowest level of control which may be thought of as a control field which changes for every record listed. F denotes final level control, the highest control level associated with a report, which operates when grand totals are printed at the end of the report.

2.1.3 Totalling fields

A maximum of ten totalling fields are allowed for the accumulation of decimal data and one of the characters 0 1 2 3 4 5 6 7 8 9 0 is used to indicate such a field.
2.1.4 Transfer fields

There may be as many as ten transfer fields and these are denoted by a character from the following set:

A B C D E G H I J K

2.2 REPORT PAGE WIDTH

Filetab reports may have an output line of up to 160 characters, the practical line length depending on the number of print positions on the line printer. The layout of the printed line is defined by parameter records where one character position is used to represent one print position. Since line printers have in excess of 80 print positions, up to two input records are required to describe a line of print.

An attempt by a COBOL program to print a line longer than the number of print positions available on the printer causes an unwanted diagnostic message. The line printer at the installation where the translation macros were developed had the popular 120 characters per line facility, so this was adopted as the maximum line width for the generated COBOL program.

The macro processor used as a basis for the COBOL generation feasibility exercise was designed for interactive terminal use and had an input buffer restricted to a maximum of 72 characters.

The combination of the above two factors made it necessary to restrict Filetab parameter records to 72 characters and to use 48 characters from a second parameter record for print positions 73 to 120.
3. DIRECTIVE AND PARAMETER FORMATS

3.1 *FILE DIRECTIVE

The *FILE directive is used to describe the nature of the input file to the system. The following format shows the options implemented in this study:

*FILE peripheral-code file-name

peripheral-code is a two character code being one of the following:
- MT Magnetic tape
- CR Card reader

file-name is the name of the input data file. The file-name is optional for card input but mandatory for magnetic tape.

N.B. If the file-name for magnetic tape is left blank the COBOL generator will assume that the file label contains blanks and will assign a default file name in the COBOL program. A maximum of 12 characters is permitted. The file-name may be terminated by a comma if it is less than 12 characters long.

3.2 *INL DIRECTIVE

This directive precedes the parameter records which associate a field specifying character with each of the required data fields. They define the way in which each field is to be processed in order to produce the desired report.

3.2.1 *INL parameter records

The entries in the parameter records are separated by commas and the next directive terminates the list. For the purposes of
this study it was assumed that no individual entry overflowed on to the next parameter record.

The entries take the form:

FSC FD

where FSC is the relevant field specifying character (Appendix II Section 2.1) and FD is the field definition which gives the start position and either length or end position of the field.

3.2.2 Field definitions

All fields in a record are referenced by their start position relative to the start of the record. The first character of the record is assigned position zero.

The Field definition may take either of the following forms:

S-E or S/L

where

S Start position of the field
E End position of the field
L Length of the field

The positions and lengths are defined in terms of character position references.

No attempt was made in this study to implement the Filetab Card Column notation and the above forms were applied to both magnetic tape and cards.

3.3 TITLE DIRECTIVE

This directive is the first of three associated with the report format specification. It precedes the parameter records containing the text of a main heading to be printed on the first page of the report.
The format of this directive is:

*TITLE b,a

where \( b \) is the number of lines to be thrown before the print line(s) and \( a \) is the number of lines to be thrown after the print line(s).

\( b \) and \( a \) are unsigned integers and either \( b \) or \( a \) may be omitted in which case the default values 1 and 3 are respectively assigned to them.

3.3.1 *TITLE parameter records

The parameter records which follow the *TITLE directive may specify any number of lines called a 'print group'. Each 120 character print line requires two input records, the first contains 72 characters and the second the remaining 48 characters. Where an odd number of records is presented the last line is assumed to contain spaces beyond the first 72 print positions.

3.4 *HEAD DIRECTIVE

This directive precedes the parameter records containing the text to be printed as a heading at the head of every page or in association with a control break. The format of the directive is:

*HEAD control-level b,a,p

where

control-level indicates the heading group which will be printed at a break in the given control field and will be a single character from the set LMNOPQR

\( b,a \) are the number of lines to be thrown before and after the page heading
p is the maximum number of lines to be printed between occurrences of the heading. In this study it is used at list level L.

b, a and p are unsigned integers.

Parameters may be omitted from right to left, intermediate parameters may not, however, be omitted. The default values assigned at missing control-level, b, a and p parameters are L, 1, 3 and 60 respectively.

3.4.1 *HEAD parameter records

The parameter records which follow the *HEAD directive are as for the *TITLE parameter records (Appendix II Section 3.3.1).

3.5 *OUT DIRECTIVE

This directive precedes parameter records giving, in picture format, an image of the print line(s) required for the main body of the report.

A parameter set must be specified for each print group to be printed at a control break, including List and Final levels.

The format of this directive is:

*OUT control-level b,a

where

control-level indicates the output group which will be printed at a break in the given control field and is a single character from the set L M N O P Q R F

b,a are the number of lines to be thrown before and after the print group has been printed. b and a are unsigned integers.

Parameters may be omitted from right to left, intermediate parameters may not, however, be omitted. The default values
assigned to the missing parameters for control-level, b and a are L, 1 and 3 respectively.

3.5.1 *OUT parameter records

In common with other directives whose parameter records specify print line formats, two records per line are used (Appendix II Section 3.3.1). Text may be included in the print line by enclosing a literal within single quotation marks. The field specifying character is entered in the record positions in which it is desired that the field contents should be printed.

3.6 *COMMENT DIRECTIVE

A Filetab comment directive consists of an * followed by a space and the desired comment. Such directives may precede any other directive.

For the purposes of this study they serve two functions in the generated program:

1. The characters from positions 3 to 6 in the first comment record are used as the program-id. A default program-id is assigned if no comment directives are included in the input.

2. The text from all comment directives is formed into a NOTE paragraph at the beginning of the Procedure division of the COBOL program.

3.7 *GO DIRECTIVE

The *GO directive terminates the Filetab parameter set and indicates to the macro definition that the generation of the COBOL program statements may be completed.
### APPENDIX III

**MACRO PROCESSING OF FILETAB DIRECTIVES**

1. **SPECIAL PURPOSE STRING AND GLOBAL VARIABLES**
2. * COMMENT DIRECTIVE
3. *FILE DIRECTIVE
4. *INL DIRECTIVE
5. *TITLE DIRECTIVE
6. *HEAD DIRECTIVES
7. *OUT DIRECTIVES
8. *GO: DIRECTIVE

<table>
<thead>
<tr>
<th>-section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>
APPENDIX III

MACRO PROCESSING OF FILETAB DIRECTIVES

This appendix contains notes on the processing used to generate COBOL statements from the Filetab directives. Listings of the macro-time statements used are included in the separate folder (Items 4 to 10). Details of the special purpose string and global variables used to carry information from phase 1 to phase 3 of the system via the CONSTANTS macro are also included. The material is presented in the following order:

1. Special purpose string and global variables
2. * comment directive
3. **FILE directive
4. *INL directive
5. *TITLE directive
6. *HEAD directive
7. *OUT directive
8. *GO directive
1. SPECIAL PURPOSE STRING AND GLOBAL VARIABLES

The following variables are used in the CONSTANTS macro to pass information from phase 1 to phase 3 of the Filetab to COBOL generating system:

- Contains the File name used in the COBOL File definition.

- Contains the Label option clause for use in the COBOL File definition.

- Contains the control level characters of those levels for which headings are to be generated.

- Switch used to indicate whether or not COBOL statements were generated for the NOTEPARA macro.

- Switch used to indicate whether or not COBOL statements were generated for the TITLEPAR macro.

- Switch used to indicate whether or not COBOL statements were generated for the FINALPAR macro.

- Switch used to indicate whether or not COBOL statements were generated for the ADDPARA macro.

- Contains the control level characters of those levels for which output is to be written.

- Contains the hardware name for the peripheral to be used for the input data file.
#GV2 - Contains the four characters to be used as the COBOL program identification

#GV2 - Contains the maximum number of lines to appear on a report page.

2. * COMMENT DIRECTIVE

This is an optional directive but, if present in a Filetab job, its contents are formed into a COBOL NOTE statement. Since in COBOL any paragraph which begins with a NOTE is taken to consist entirely of commentary, a special NOTE-PARA paragraph is created at the beginning of the Procedure division for this purpose.

The first comment directive initiates the generation of the NOTE-PARA paragraph name and the NOTE statement. The characters in positions 3 to 6 of the * comment record are used to override the default values set for the program identification. A global variable #GS01 is set equal to 1 to denote the presence of a NOTE statement. Global variable #GV2 is used to store the program identification.

The comment statement may consist of up to 72 characters, which is longer than the space beyond the B margin of a COBOL statement. It is therefore arbitrarily broken into two COBOL lines, one of 60 characters and the other of 12. No attempt is made to interpret the contents of the comment records and this may lead to broken words and blank comment lines within the COBOL program.

Use is made of the routines which generate the system macro statements for filing the generated COBOL statements in the required subfile, which in this case is called NOTEpara.
When the Filetab comment record has been processed and the appropriate statements generated, control is passed to the statement which reads in the next Filetab directive, i.e. the macro-time statement with 1 as its label.

The listing of macro-time statements for processing the *comment directive appears as item 4 in the separate folder.

3. *FILE DIRECTIVE

The *FILE directive contains details of the peripheral-code and file-name. The file-name is optional for cards but mandatory for magnetic tape, although in this case blanks are permitted. The peripheral-code is a two-character code in positions 7 and 8 of the *FILE record and the file-name, if present, occupies positions 10 to 21. A comma may be used to terminate the file-name if it is less than 12 characters long.

The generator uses the Filetab file-name, if present, as both the file-name and file label identification in the File description statements in the File section of the COBOL program's Data division. If the Filetab file-name is omitted, the generator assigns DEFAULT-NAME as the file-name for use in the COBOL program.

When the Filetab file-name is omitted and the peripheral-code is for card input, the COBOL program will specify that Label records are omitted. On the other hand, if the file-name is omitted and the Filetab peripheral-code is for magnetic tape, the COBOL program will specify that Label records are standard but that the File label identification contains blanks.

File details are also required in the Environment division, Input-Output section, File-Control paragraph and the Procedure division paragraphs which contain the OPEN, READ and CLOSE verbs, so special purpose global and string variables are used to store the information derived from the *FILE directive.
While Filetab uses 'CR' and 'MT' to indicate the peripheral codes, the corresponding COBOL hardware names used in the ASSIGN clause of the File-Control paragraph are 'CARD-READER' and 'TAPES'. Since a macro-time variable can hold a maximum of 8 characters, two Global variables, #GV11 and #GV12, are used to store the hardware names. The way in which the hardware-name characters are assigned to the Global variables is shown below.

<table>
<thead>
<tr>
<th>Filetab Peripheral-code</th>
<th>Contents of macro processor Global variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#GV11</td>
</tr>
<tr>
<td>CR</td>
<td>CARD-READER</td>
</tr>
<tr>
<td>MT</td>
<td>TAPE</td>
</tr>
</tbody>
</table>

An invalid peripheral-code will cause the COBOL generating macro to print an error message.

String variable #506 is used by the generating macro to hold the file-name for use in the Environment, Data and Procedure divisions of the generated program. The Label record clause for use in the File definition of the generated program is deposited in String variable #507.

After the data from the *FILE directive has been extracted and interpreted, the macro generates the macro-time statements which save the contents of #506 and #507. These generated macro-time statements are 'grown' into the macro definition CONSTANTS, which is executed in the third phase of the system in order to restore these same values to the String variables #506 and #507.

When the *FILE directive has been processed and the constants set up, control is transferred to the statement which reads the next Filetab directive (See Figure 3.6).

The listing of macro-time statements for processing the *FILE directive appears as item 5 in the separate folder.
4. *INL DIRECTIVE

To assist with the generation of the REDEFINES statements used for each new record description, a field count variable $GFCT is maintained. This is initially set equal to zero.

The parameter records are read and processed one at a time until a new Filetab record, which begins with the * character, is detected. After each read operation the reserved global variable $GINP, used for pointing at a character in the macro processor input buffer, is set equal to zero.

Each field description is terminated by a comma or the end of the parameter record and it is assumed that no field specification continues on to another parameter record. The whole parameter record is squashed to remove unwanted spaces and the number of useful characters is automatically set in reserved variable $GSTC. A copy of the $GSTC value is stored in local variable $LSTC, because subsequent READ operations, which occur during the manipulation of some strings of numeric data, cause the value of $GSTC to be set equal to zero.

Each field description is processed character by character in order to extract the four constituent parts. Global variable $GV4 is used to hold the field specifying character, $GV5 is used to hold the start position of the field and $GV7 is used to hold the end position or length of the field. A '-' character stored in $GV6 denotes that end position is being specified, while a '/' character is used to denote that the length has been specified.

The CALLVSTR macro-time facility for extracting variable length character strings depends on the recognition of a specific terminator character. Since the terminator for the start position and length or end position strings has two alternatives, this facility cannot be used. This problem suggests that the development of an alternative form of the CALLVSTR function could
be considered, in which the string extraction is terminated when a match with any one of a list of characters is encountered. To circumvent the problem, the data is extracted character by character and stored in string variable #S01 until a non-digit character is detected. These strings are then transferred to the global variables #GV5 and #GV7 using the macro processor input buffer. Another string variable #S09 is used as temporary storage for the complete Filetab parameter record. Before arithmetic operations can be carried out on the numeric data in #GV5 and #GV7, they have to be converted from characters to binary using the CALLCONV macro-time statement.

The start and end position option, denoted by a '-' character in #GV6, shows that the values in #GV5 and #GV7 refer to card columns which have 1 as the origin. In contrast, the start and length option, denoted by a '/' character in #GV6, refers to records on other media where 0 is used as the origin. When the former option is used in the Filetab parameter record, the values in #GV5 and #GV7 are recomputed to reflect the 0 origin and the field length.

The File section COBOL statements for defining the input record (Section 3.5 of main text) were specifically designed to have a group structure so that the REDEFINES facility could be used at the 02 level. In the case of the first field on the Filetab parameter record, this takes the form

```
02 FILLER-1 REDEFINES IN-REC.
```

Subsequent fields make use of the field counter #GFCT to generate statements of the type

```
02 FILLER-n REDEFINES FILLER-m.
```

where m = n - 1 and n is the current value in #GFCT.

The level 03 statements for each record description are generated in three stages:
1. The initial FILLER field(s), if any.
2. The actual data field.
3. The final FILLER field(s), if any.

The value in GV5 is used to generate the initial FILLER field(s), if any. If the required filler exceeds 120 characters, the maximum permitted in ICL 1900 COBOL, it is subdivided into multiples of 120 characters and a remainder. Since a divide operation is not available in the PG/1 macro processor facilities, this calculation is carried out by repeated subtraction.

The field specifying character in GV4 and the field length in GV7 are used to generate the actual data field level 03 statement. In the case of a totalling field, the numeric field specifying character has the letter A prefixed in order to make it a valid data name in COBOL, and the numeric picture character 9 is used with the value in GV7 as the repetition count.

```
03 A#GV4 PICTURE 9(#GV7).
```

Transfer and control break fields use the alphabetic field specifying character from GV4 with the alphanumeric picture character X. Again the value in GV7 is used as the repetition count.

```
03 #GV4 PICTURE X(#GV7).
```

The length of the final filler field is calculated as

```
#LV1=2048-#GV5-#GV7
```

and the FILLER statement(s), if any, are generated in a similar way to the initial ones.

In the case of totalling fields, the field specifying character in GV4 is used to generate the statements required to define the control totals in the Data division and to actually accumulate them in the Procedure division. Thus the statements

```
03 A#GV4 PICTURE 9(15) COMPUTATIONAL VALUE ZERO.
```

and
ADD A\#GV4 IN INREC TO A\#GV4 IN M-LEVEL.

are used to generate the COBOL statements for inclusion in the SAVETOTL and ADDPARA macros respectively.

The control break field, whose field specifying character is one of the set MNOPQR, makes use of the values in \#GV4 and \#GV7 to generate the COBOL statement for inclusion in the Data division group structure used to detect sequence changes.

O2 \#GV4 PICTURE X(\#GV7).

The generated statement is destined for inclusion in the CTRLBRKE macro.

In all cases the necessary filing statements are also generated so that the macros to which they contribute can be 'grown'.

The macro-time statements for processing the *INL directive appear as item 6 in the separate folder.

5. *TITLE DIRECTIVE

The *TITLE directive and its associated parameter records specify the Report title to be printed at the top of the first page of output. This directive is optional. The *TITLE directive contains details of the line spacing required before and after the Report title. Default values for the line spacing are set if the second or both spacing parameters are omitted.

The directive is followed by parameter records which contain the text of the heading. Each 120 character line of text is entered on a pair of parameter records; the first contains the first 72 characters of the print line while the second contains 48 characters to be printed in positions 73 to 120 of the line. If the last 48 characters of the last line of the title are all to be blank, then the second record of the final pair may be omitted.

The information extracted from the *TITLE directive and
parameter records is required at two points within the generated COBOL program. In the Data division a data description for each line of the Report title is required, while the Procedure division must contain the WRITE statement(s) necessary to output the title line(s) with the required spacing. The COBOL statements are therefore directed to two macro definition subfiles, TITLES and TITLEPAR, by making use of the routine which generates the system macro filing statements.

The AFTER ADVANCING option of the WRITE verb is used to effect the required spacing before the first line of the Report title. Subsequent title lines are single spaced before printing. The final spacing after the Report title is achieved by writing a blank line with the desired spacing.

The processing of the *TITLE directive currently in the input buffer begins by setting global variable $GS03 equal to 1 to indicate the presence of this optional directive. Global variables $GV4 and $GV5 are used by this routine to contain the line spacing counts before and after printing. Initially these are set to have default values of 1 and 3 respectively.

The spacing options, if present, are extracted from the directive record and the title line counter, global variable $GV6, is initially set to 1. This latter variable is used to generate a level 01 group data description entry for each pair of parameter records, which takes the following form

01 TITLE-n.

where n is the value in $GV6. The value in $GV6 is incremented each time the first of a new pair of parameter records is detected.

The text from each pair of parameter records is extracted and used to generate three level 02 FILLER literals within the title line group description. The data from the first parameter record is used to generate two literals of 58 and 14 characters, while the
second parameter record's data is used to generate a 48 character literal to complete the record description. All the above statements are destined for the TITLES macro definition subfile.

The Procedure division statements generated by this routine are filed in the TITLEPAR macro definition subfile and are concerned with the actual writing out of the report title data. The contents of #GV4 are used in the generation of the WRITE statement for the output of the first title line. Subsequent title lines are single spaced but the contents of #GV6 are used to denote the actual record to be output. When all pairs of parameter records have been processed, the contents of #GV5 are used in the generation of statements to output the required number of blank lines after the title. In all cases statements to update the COBOL program LINE-COUNT data item are also generated.

An asterisk in position 1 of the first of a pair of parameter records is the signal that the *TITLE directive has come to an end and another directive has been detected. Control is then passed to the section of processing which identifies the new directive (See Figure 3,6). Listing Item 7 in the separate folder shows the macro-time statements for processing the *TITLE directive.

6. *HEAD DIRECTIVES

These directives and associated pairs of parameter records specify the text to be printed as a heading for each control level at the head of every page or in association with a control break. Each directive specifies the control level field to which the heading applies, the line spacing before and after the heading and in the case of control level L the maximum number of lines to be printed between occurrences of the heading.

As with the *TITLE directive, default values are set for the various options which take effect when they do not specifically
appear on the directive records. Global variables $GV4, $GV5, $GV6 and $GVZ are used to contain the control level character, the line spacing counts before and after printing and the maximum number of lines value respectively. The pairs of parameter records containing the heading text are processed in a similar way to those for the "TITLE" directive.

A count of lines within each control level heading is maintained in global variable $GHCT and this, together with the control level character, is used to define the level 01 group entry in the COBOL program Data division for each heading line,

```
01 X-HEAD-n.
```

where X is the control level character from $GV4 and n is the line count value within that level from $GHCT. These group structures are generated for inclusion in the macro definition subfile called HEADINGS.

The Procedure division WRITE statement(s) to output the heading lines with the required spacing are generated in a similar way to those for the Report title. Global variables $GV5 and $GV6 are respectively used to effect the desired spacing before the first heading line and the blank line spacing after the heading. Global variable $GHCT denotes the number of the line within the current heading and this is used to specify the name of the record to be written out. As before, statements to update the COBOL program LINE-COUNT data item are generated. All the Procedure division statements associated with a control level heading are generated together with the necessary system filing statements for inclusion in the OTHERPAR macro definition.

An asterisk in position 1 of the first of a pair of parameter records signals the end of the current "HEAD" directive and control passes to the macro-time statements which identify the new Filetab directive (See Figure 3.6 in main text).
The control level character from each *HEAD directive present in the Filetab job specification being processed is stored in the global variable %GHED, which is later used during the processing associated with the *GO directive.

A listing of the macro-time statements for processing the *HEAD directives appears as item 8 in the separate folder.

7. *OUT DIRECTIVES

The *OUT directives and their associated parameter records specify the contents and format for the detail and control total line(s) for each level of the report. Each directive specifies the control level to which the format set out in the parameter records applies, together with the line spacing before and after the output lines. The processing of the *OUT directive record is similar to that for the *HEAD and *TITLE directives in that default values are set for any options not specifically coded. Global variable %GOCT is used to maintain a count of lines within a print group and global variables %GV4, %GV5 and %GV6 are used to store the control level character and the line spacing counts before and after the print group.

The Data division output record descriptions are generated for inclusion in the OUTREC subfile and the level 01 entry takes the form

    01 X-RECh.

where X is the control level character from %GV4 and n is the current line count value in %GOCT.

The WRITE statements which are generated for each level of the report from the information contained in the *OUT directive differ in three ways from those generated from the *TITLE and *HEAD directives. First, those generated for the detail level, L, are filed in the WRITEPAR macro definition subfile while those for
the other levels are filed in the OTHERPAR subfile. Secondly, the first WRITE statement of a group for any control level, excluding L, is preceded by a Procedure division paragraph name which takes the form

X-TOTALS.

where X is the control level character in $GV4. Finally, before the first WRITE statement for any print group a PERFORM statement is generated. This latter statement ensures that a new report page is started if the line count has reached the maximum permitted value as specified in the level L *HEAD directive, if present, otherwise the default value is used.

The pairs of parameter records which follow the *OUT directive are different from those of the two previously discussed directives. They specify not only text information but give details of the data fields (transfer, control or totalling) which are to appear in the printed line.

Each pair of parameter records which specify the format and contents of a report line is scanned character by character to establish the type and size of the fields contained therein. Any one of four different types of level 02 entry may have to be generated for inclusion in the COBOL group description for a record. Blank characters in the line parameter records cause the generation of a blank alphanumeric FILLER field with length equal to the number of consecutive blank characters, e.g.

02 FILLER X(n) VALUE SPACES.

where n is the count of consecutive blank characters which is accumulated in global variable $GV7. A filetab transfer or control character string in the parameter record(s) causes the generation of an alphanumeric data item of the following form:

02 Y PICTURE X(n).

where Y is the Filetab field specifying character stored in global
variable #GV9 and n is the length of the field. This letter is the number of consecutive occurrences of the field specifying character and this is again accumulated in #GV7. A Filetab totalling field specification causes the generation of a zero suppressed numeric report item which takes one of the following forms depending on the field width:

    02 AY PICTURE Z(n-1)9.
    02 AY PICTURE 9.

Y and n have the same meanings as above. Finally, a Filetab text literal which is enclosed in single quotation marks (') causes the generation of a COBOL alphanumeric FILLER field with a VALUE clause. The value assigned to the COBOL literal is the same as the Filetab literal together with one leading and one trailing blank character which replace the unused print positions occupied by the single quotation marks. String variable #$09 is used to store the Filetab literal. It should be noted incidentally that ICL 1900 COBOL uses the double quotation character (" ) to delimit a literal value. All 02 level descriptions are generated for inclusion in the OUTREC macro definition subfile.

In addition to the above record description entries, it is also necessary to generate MOVE statements to set up the required values in the record descriptions prior to the execution of the WRITE statements. In the case of the *OUT directive at control level L, these take the form

    MOVE X IN INREC TO X IN L-RECn.

where X is the COBOL data name of the Filetab field and n is the record number within the level L output print group. These statements are generated for inclusion in the MOVEPARA macro definition subfile. At all other control levels of the *OUT directive, the MOVE statements take the form

    MOVE CORRESPONDING X-LEVEL TO X-RECn.
where \( X \) is the control level character and \( n \) is the record number within the output print group for that level. These latter statements are generated for inclusion in the OTHERPAR macro and precede the WRITE statement for the record to which they refer.

An asterisk in position 1 of the first of a pair of parameter records signals the end of the current "OUT directive and control passes to the macro-time statements which identify the new Filetab directive (Figure 3.6 of the main text).

The control level character from each "OUT directive present in the Filetab job specification being processed is appended to the contents of global variable #OUT, which is later used during the processing associated with the "GO directive.

A listing of the macro-time statements for processing "OUT directives appears as item 9 in the separate folder.

8. "GO DIRECTIVE

The "GO directive has no parameters and for the purposes of the COBOL program generating system merely serves to terminate a set of Filetab input data. However, once this directive is detected, there are seven tasks which may have to be carried out in order to generate statements needed to complete the macros being 'grown' by the system.

1. The first task is only carried out if a * comment directive was present in the Filetab input data. It consists of generating a full stop to terminate the NOTE statement in the NOTE-PARA paragraph.

2. The second task concerns the generation of the COBOL statements to ensure that the headings desired for each control level are output at the top of each new page. This task falls into two parts, first the generation of a PERFORM statement and secondly the generation of the paragraphs to be performed.
The writing out of the control level heading is executed by means of the PERFORM verb because the same processing is also required when a control sequence break is detected. The order in which the control levels are processed ranges from List level L through the major to the minor control level as contained in the global variable $GHED. Thus for control level M the statement

```
PERFORM M-HEAD THRU M-HEAD-EXIT.
```

would be generated for inclusion in the PAGEPARA macro. The following paragraphs would also be generated for inclusion in the OTHERPAR macro definition subfile.

```
M-HEAD.
PERFORM PARA-LINE-CHECK THRU PARA-LINE-EXIT.
IF PAGE-SWITCH NOT EQUAL TO ZERO GO TO M-HEAD-EXIT.
PERFORM M-HEAD-WRITE.
M-HEAD-EXIT. EXIT.
```

3. The next task is concerned with the final processing in the COBOL program when the end of file on the input data file is detected. The final totals for all the control levels used in the job must be processed and written out in minor to major control level sequence. As before, this falls into two parts: first, the generation of the necessary PERFORM statements for inclusion in the FINALPAR macro definition subfile; and secondly, the generation of the paragraph which causes the current level control totals to be added to the next higher level totals. Because this same paragraph is also performed during the normal processing of a control break sequence, it contains a statement which resets the totals for the current control level to zero. For example, in a Filetab job using control levels M and N, the following statements are generated for control level M, the first statement is for inclusion in the FINALPAR macro definition subfile and the remaining statements are for the OTHERPAR subfile.

```
PERFORM M-TOTALS.
M-ADD.
ADD CORRESPONDING M-LEVEL TO N-LEVEL.
SUBTRACT CORRESPONDING M-LEVEL FROM M-LEVEL.
```
4. The generation of the COBOL statements for detecting and processing a control sequence break forms the next task. These statements form a complete set of paragraphs, beginning with PARA-BREAK and ranging through to PARA-BREAK-EXIT, which are generated for inclusion in the BREAKPAR macro definition subfile. The IF statement which detects the sequence break is followed by statements which cause the control totals for all lower levels and for the current control level to be printed. These are then followed by statements which update the control break key fields and cause new headings for the current and lower control levels to be printed. If the previous example of a Filetab job with control levels M and N is used again, then the following paragraphs will be generated.

```
PARA-BREAK.
  IF N IN INREC EQUALS N IN CONTROL-BREAK
  GO TO M-BREAK.
  PERFORM M-TOTALS.
  PERFORM N-TOTALS.
  MOVE N IN INREC TO N IN CONTROL-BREAK.
  MOVE M IN INREC TO M IN CONTROL-BREAK.
  PERFORM N-HEAD THRU N-HEAD-EXIT.
  PERFORM M-HEAD THRU M-HEAD-EXIT.
  GO TO PARA-BREAK-EXIT.
M-BREAK.
  IF M IN INREC EQUALS M IN CONTROL-BREAK
  GO TO PARA-BREAK-EXIT.
  PERFORM M-TOTALS.
  MOVE M IN INREC TO M IN CONTROL-BREAK.
  PERFORM M-HEAD THRU M-HEAD-EXIT.
PARA-BREAK-EXIT. EXIT.
```

5. The generation of the statements which enable the PART1 macro to be 'grown' forms the next task. This macro contains the Steering lines statements which specify the compiler options to be invoked for the COBOL program. The only variable data in this macro is the Identity which is contained in global variable $GV2.

6. As the penultimate task, the macro time statements for the CONSTANTS macro definition subfile are generated. This macro enables the contents of several global variables to be passed between the first and third phases of the system. The CONSTANTS macro is executed immediately prior to the GOPART2 macro which
synthesises the complete COBOL program. Thus the required contents of the global variables are restored and ready for use in the GOPART2 macro.

7. The final task is to generate the END system macro statements which will be filed to complete each of the incomplete macro definitions that have been 'grown' by the generating system.

A listing of the macro time statements for carrying out the above tasks appears as item 10 in the separate folder.
## APPENDIX IV

### PG/2 MACRO PROCESSOR

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>NEW MACRO-TIME STATEMENTS</td>
</tr>
<tr>
<td>2.</td>
<td>ADDITIONAL RESERVED GLOBAL VARIABLES</td>
</tr>
<tr>
<td>3.</td>
<td>NEW SYSTEM MACRO</td>
</tr>
<tr>
<td>4.</td>
<td>ADDITIONAL ERROR MESSAGES</td>
</tr>
<tr>
<td>5.</td>
<td>ADDITIONAL ARITHMETIC OPERATORS</td>
</tr>
</tbody>
</table>
APPENDIX IV

PG/2 MACRO PROCESSOR

The material in this appendix details the enhancements to the PG/1 macro processor for implementation in the PG/2 version and is subdivided into the following sections:

1. New or enhanced macro-time statements.
2. Additional reserved global variables.
3. New system macro.
4. Additional error messages.
5. Additional arithmetic operators.

The implementation of these facilities did not form part of the candidate's work.
1. NEW MACRO-TIME STATEMENTS

The following macro-time statements supplement or replace those listed in Section 7 of Appendix I.

1.1 CALL

Branch to the label specified and store in a private stack the return address, which is that of the next sequential instruction. The CALL statement is used to enter a subroutine which may itself contain CALL statements (Section 4.3 of main text).

% CALL label

where label is the label of the first executable macro-time statement of the subroutine. Return from the subroutine is effected by an EXIT macro-time statement (Appendix IV Section 1.10).

e.g. % CALL 9000

1.2 CALL DPOS

Concatenate strings in a macro-time variable (Section 4.6.4 of main text).

% CALL DPOS <macro-time variable>,

{<text>|<macro-time variable>}

The value of the right hand argument is concatenated with the value of the left hand argument and the result assigned as the new value in the left hand argument. Any variables referred to must have string values and the result is truncated to 8 characters in length.

e.g. % CALL DPOS (#LV1,#GV2)

If #LV1 and #GV2 initially contained 'HAPPY' and 'DAYS' respectively then #LV1 will contain 'HAPPYDAY'.

1.3 CALL LENV

Determine the length of a string or macro-time variable
containing a string value (Section 4.6.7 of main text).

\% \text{CALL LENV \{\text{macro-time variable}\}\{\text{string variable}\}\}

Global variable \#GSTC is set equal to the length of the string or macro-time variable.

e.g. \% \text{CALL LENV \#LV2}
If \#LV2 contains 'VALUE' then global variable \#GSTC is set equal to 5.

1.4 CALL SQUA

Remove spaces from string valued variable.

\% \text{CALL SQUA \langle variable\rangle}

\langle variable\rangle ::= \text{string variable}\mid \text{macro-time variable}

All spaces are removed from the specified variable if it has a string value, otherwise an error message is generated and the macro processor halts. Global variable \#GSTC is given an integer value corresponding to the number of non-space characters in the variable.

e.g. \text{CALL SQUA \#LV1}
If \#LV1 originally contained 'A B C' then it will contain 'ABC' after the execution of the statement and \#GSTC will have the value 3.

1.5 CALL SWAP

Interchange characters in a macro-time variable (Section 4.6.5 of the main text).

\% \text{CALL SWAP \langle macro-time variable\rangle,\langle character\rangle,\langle character\rangle}

Single or multiple occurrences of the left-hand character are replaced by single occurrences of the right-hand character. \#GSTC is given an integer value equal to the number of characters in the result.

e.g. \text{CALL SWAP \#LV1,B,X}
If #LV1 initially contained 'ABBE' then its contents will become 'AXE' after the execution of the statement and #GSTC will have the value 3.

1.6 CHAIN

Load a subfile and re-initialise. (Section 4.6 of the main text).

[CHAIN<control code>,<subfile name>]

The stacks and variables are re-initialised according to the option specified in the control code and the named subfile is loaded. The execution of this statement is deferred until the second pass of the macro in which it occurs. The CHAIN statement is always followed by one or more calls to the macros in the named subfile (Section 4.2 of main text).

Control code 0 = initialise stacks
1 = initialise global variables and stacks
2 = initialise string variables and stacks
4 = initialise global and string variables and stacks

e.g. [CHAIN0,FILE1]
[ M2 ]

 Initialise the stacks, load the macros in the FILE1 subfile and call the M2 macro.

1.7 DATE

Obtain the current date from the operating system executive (Section 4.7.1 of main text).

% DATE

Global variable #CDT will then contain the date in character form (Appendix IV Section 2).
1.8 DELETE

Delete subfile record (Section 4.4 of main text).

% DELETE <number>|<macro-time variable>

Delete the specified record from the currently selected subfile.

E.g. % DELETE #LREC

If #LREC is equal to 5 the fifth record of the currently selected subfile will be deleted.

1.9 DEPOSIT

Extension of deposit facility (Section 4.6.8 of main text).

% DEPOSITn <choice>

The deposit facility (Appendix I Section 5.8) is extended so that <choice> may also include <argument reference>.

E.g. % DEPOSIT3 @2@

Copy all the information in the second argument of the macro definition in which the statement occurs into string variable #303 starting at the first available 'unfilled' character position of the string variable.

If the value for n is omitted the input buffer is used as the receiving string.

1.10 EXIT

Subroutine exit (Section 4.3 of main text).

% EXIT

Branch to the return address at the top of the stack and delete this entry from the stack. This statement is used to exit from a subroutine entered by the CALL statement (Appendix IV Section 1.1).
1.11 FREAD

Read subfile record (Section 4.4 of main text).

% FREAD \langle arg1\rangle,\langle arg2\rangle
\langle arg1\rangle::=\text{number}\mid\text{macro-time variable}
\langle arg2\rangle::=\text{macro-time variable}\mid\text{string variable}

From the currently selected subfile read the record specified by the first argument into the variable specified as the second argument.

E.g. % FREAD #LREC,#S01
If #LREC is equal to 4 then the fourth record of the currently selected subfile is read into string variable #S01.

1.12 FSTR

Fixed length string extraction (Section 4.6.1 of main text).

% FSTRm #Sn,\{<macro-time variable>\mid\langle integer\rangle\},
\{<macro-time variable>\mid\langle integer\rangle\}

This statement has the same function as CALLFSTR (Appendix I Section 7.4) except that the operation is performed on string n and the result is put in string m. If #Sn is omitted then the output buffer is used as the source string. If m is omitted the input buffer is used as the destination string.

E.g. FSTR2 #S01,1,10
Extract the first 10 characters from string variable #S01 and place them in string variable #S02.

1.13 IF

Extension of the IF statement facilities (Section 4.6.2 of main text).

The operators permitted with string expressions shown in Section 5 of Appendix I are extended as follows:
SOPR ::= EQ | NE | IN | AT | LE | LT | GE | GT

The new operators are shown below:

LE  less than or equal to
LT  less than
GE  greater than or equal to
GT  greater than

1.14 INSERT

Insert record in subfile (Section 4.4 of main text).

% INSERT <arg1>,<arg2>

<arg1>::= number | macro-time variable
<arg2>::= 'actual string' | macro-time variable | string variable

In the currently selected subfile, insert before the record indicated by the first argument the insertion record contained in the second argument.

e.g.  % INSERT 3,#S03

Insert the record contained in variable #S03 before the third record of the currently selected subfile.

1.15 MAINFILE

Set the name of the Mainfile to be used by the PG/2 macro processor (Section 4.7.3 of main text).

% MAINFILE,<mainfile name>

This statement is used to change the name of the mainfile when it is desired to reference a subfile not in the current mainfile.

1.16 PRINT

Empty the contents of the output stack on to the PG/2 output device (Section 4.5 of main text).

% PRINT
1.17 REPLACE

Replace subfile record (Section 4.4 of main text).

% REPLACE <arg1>,<arg2>

<arg1>: = number|macro-time variable
<arg2>: = macro-time variable|string variable

In the currently selected subfile, replace the record indicated by the first argument by the replacement record contained in the second argument.

e.g. % REPLACE #LREC,#S01

Replace the record indicated by the value in #LREC by the record contained in string variable #S01.

1.18 SAVE

Empty the contents of the output stack on to the specified subfile (Section 4.5 of main text).

% SAVE,<control code>,<subfile name>

Control code 0 = store the stack contents starting at the beginning of the named subfile

1 = append the stack contents to the end of the named subfile

N.B. This statement supercedes the statement with the same name in the PG/1 version (Appendix I Section 7.21).

e.g. % SAVE,0,SELC0ND

Empty the output stack and store the contents at the beginning of the SELCOND subfile.

1.19 SELECT

Select subfile (Section 4.4 of main text).

% SELECT <arg>

<arg>: = subfile name|macro-time variable|string variable

Set the name of the subfile to which subsequent REPLACE, INSERT,
DELETE and FREAD statements will refer.

e.g.  %  SELECT CATALOGUE

Select the CATALOGUE subfile.

1.20 SQUASH

Extension of squash facility (Section 4.6.6 of main text).

%  SQUASHn

The squash facility (Appendix I Section 7.22) is extended so that
the operation is performed on string variable n. If n is omitted
the operation is performed on the input buffer as before, but if
n is equal to 0 the operation is performed on the output buffer.
The number of characters in the result is stored in global
variable #GSTC.

e.g.  %  SQUASH3

If #SO3 initially contained 'A B C D' then its contents would
become 'ABCD' and #GSTC will have the value 4.

1.21 SWAP

Interchange characters in a string variable (Section 4.6.5
of main text).

%  SWAPn,<character>,<character>

Single or multiple occurrences of the left-hand character in string
variable n are replaced by single occurrences of the right-hand
character. If n is omitted the input buffer is used as the source
string and if n is equal to 0 the output buffer is used as the
source string. Global variable #GSTC is given an integer value
equal to the number of characters in the resulting string.

e.g.  %  SWAP1,B,X

If #SO1 initially contained 'AABBCCDD' then it will contain
'AAXCCDDE' after the execution of the statement and #GSTC will
1.22 TIME

Obtain the current time from the operating system executive (Section 4.7.1 of main text).

% TIME

Global variable $GCTM$ will then contain the time in character form (Appendix IV Section 2).

1.23 VSTR

Variable length string extraction (Section 4.6.1 of main text).

% VSTRm $\#n$, {<macro-time variable>|<integer>},
   {<macro-time variable>|<integer>}

This statement has the same function as CALLVSTR (Appendix I Section 7.5) except that the operation is performed on string $n$ and the result is put in string $m$. If $\#n$ is omitted then the output buffer is used as the source string. If $m$ is omitted the input buffer is used as the destination string.

e.g. ... % VSTR1 $\#S03,4,X$

Extract characters from string variable $\#S03$ starting with the fourth character until an X character is detected and place them in string variable $\#S01$. 
2. ADDITIONAL RESERVED GLOBAL VARIABLES

The following variables are reserved within the PG/2 macro processor system and are additional to those listed in Section 8 of Appendix I. Their use and the statements which affect each are indicated.

CDT - (stands for Character form of Date) holds the date as eight characters in the form DD/MM/YY where:

DD represents a two digit day number

MM represents a two digit month number

YY represents a two digit year number

The value of this global variable is set during the execution of the DATE macro-time statement (Appendix IV Section 1.7).

CTM - (stands for Character form of Time) holds the time as eight characters in the form HH/MM/SS where:

HH represents a two digit hour number

MM represents a two digit minute number

SS represents a two digit second number

The value of this global variable is set during the execution of the TIME macro-time statement (Appendix IV Section 1.22).
3. NEW SYSTEM MACRO

%MAINFILE

The form of this command is:

%MAINFILE,(mainfile name)

It causes the name of the mainfile referenced by the PG/2 macro processor to be changed (Section 4.7.3 of main text). The PG/2 macro processor contains a built-in default value for the mainfile name, but this command causes the default value to be overwritten by the specified mainfile name.

E.g. %MAINFILE,NEXTFILE

4. ADDITIONAL ERROR MESSAGES

The following error number messages replace or supplement those shown in Section 9 of Appendix I.

71 - Now applies to all file handling errors detected at the execution stage.

76 - In addition to the error indicated in Section 9 of Appendix I, this error number now applies to all errors detected in the new macro-time statements at the compilation stage.

77 - This applies to all errors detected in the new PG/2 macro-time statements at the execution stage.

5. ADDITIONAL ARITHMETIC OPERATORS

The PG/1 macro processor range of simple arithmetic operations between integers and variables holding integer values (Appendix I Section 5) is extended to include multiplication and division. The operators for multiplication and division are denoted by the characters * and / respectively.
APPENDIX V

FORMATS OF COMMON DATA SUBFILES

1. CATALOGUE SUBFILE
2. RELATIONAL SUBFILE
3. SUBMODEL SUBFILE
4. DOMAINDICT SUBFILE
5. KEYDICT SUBFILE
6. DOMAINLIST SUBFILE
7. DOMAININDEX SUBFILE
8. LABELTABLE SUBFILE
9. SUBFILES WHOSE NAMES HAVE THE PREFIX PAGE
10. PICTUREDICT SUBFILE
APPENDIX V

FORMATS OF COMMON DATA SUBFILES

The format details of all the PG/2 macro processor common data subfiles used by the COBOL report program generating system are given in the order in which they are used. Where a subfile contains more than one record type the record formats are presented in the order in which they occur in the subfile. In records containing more than one field the comma is used as the field separator character.

The formats of the following subfiles are described:

1. CATALOGUE
2. RELATIONDICT
3. SUBMODEL
4. DOMAININDICT
5. KEYDICT
6. DOMAINLIST
7. DOMAININDEX
8. LABELTABLE
9. PAGEname
10. PICTURESICICT
## 1. CATALOGUE SUBFILE

### 1.1 CONTROL RECORD

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No. of File descriptions in the Catalogue</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>No. of records in the Catalogue</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Catalogue Password</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td>Any characters including blanks. Only used when adding or deleting file descriptions. Not used for read access.</td>
</tr>
<tr>
<td>4</td>
<td>No. of levels of security protection for read access to fields</td>
<td>N</td>
<td>V</td>
<td>2</td>
<td>Current maximum is 12, but may readily be increased to suit user's requirements.</td>
</tr>
</tbody>
</table>
| 5         | Date of last Catalogue update                 | A    | F      | 8         | DD/MM/YY where
DD = day number  
MM = month number  
YY = year number                                         |
| 6         | Time of last Catalogue update                 | A    | F      | 8         | HH/MM/SS where
HH = hours  
MM = minutes  
SS = seconds                                             |

Type: A = Alphanumeric, N = Numeric  
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 42
1.2 FILE NAME RECORD TYPE 1

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File name</td>
<td>A</td>
<td>V</td>
<td>30</td>
<td>Must obey the rules for COBOL data-names.</td>
</tr>
<tr>
<td>2</td>
<td>No. of fields described in the Catalogue</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td>Not all fields in the file need be included in the Catalogue.</td>
</tr>
<tr>
<td>3</td>
<td>Password-1</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td>Highest security level.</td>
</tr>
<tr>
<td>4</td>
<td>Password-2</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td>Security levels are entered in decreasing order.</td>
</tr>
<tr>
<td>5</td>
<td>Password-2</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td>Used only if three or more levels of security are used in the Catalogue.</td>
</tr>
<tr>
<td>6</td>
<td>Password-4</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td>Used only if four or more levels of security are used to protect the Catalogue data.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 72
1.3 FILE NAME RECORD TYPE 1C

This record is present only if the number of security levels used to protect the Catalogue is greater than 4.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Password-5</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Password-6</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Password-7</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Password-8</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Password-9</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Password-10</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Password-11</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Password-12</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Only as many fields as are required by the security level system are present. Fields are in decreasing order of data sensitivity.

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 72
### FILE NAME RECORD TYPE 2

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. Size</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Description of file contents</td>
<td>A</td>
<td>V</td>
<td>70</td>
<td>The contents of this field is entirely at the user's discretion. A field separator character is not used to terminate this field.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric  
Length: V = Variable  

Maximum record size = 70 characters
### 1.5 FILE NAME RECORD TYPE 3

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File medium</td>
<td>A</td>
<td>F</td>
<td>2</td>
<td>CR = cards, FT = paper tape, MT = magnetic tape, ED = exchangeable disc.</td>
</tr>
<tr>
<td>2</td>
<td>Maximum block size</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td>$1 \leq \text{Block size} \leq 2048$</td>
</tr>
<tr>
<td>3</td>
<td>Maximum record size</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td>$1 \leq \text{Record size} \leq \text{Maximum block size}$</td>
</tr>
<tr>
<td>4</td>
<td>File label</td>
<td>A</td>
<td>F</td>
<td>12</td>
<td>Only used for MT and ED files.</td>
</tr>
<tr>
<td>5</td>
<td>Access mode</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>Used only for ED files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$S = \text{sequential}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$R = \text{random}$</td>
</tr>
<tr>
<td>6</td>
<td>Organisation</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>Used only for random access ED files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$D = \text{direct}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$I = \text{indexed}$</td>
</tr>
<tr>
<td>7</td>
<td>Length of symbolic key</td>
<td>N</td>
<td>V</td>
<td>2</td>
<td>Used only for random access ED files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$1 \leq \text{Length} \leq 64$</td>
</tr>
<tr>
<td>8</td>
<td>No. of fields forming the</td>
<td>N</td>
<td>V</td>
<td>1</td>
<td>$1 \leq \text{Number} \leq 9$</td>
</tr>
<tr>
<td></td>
<td>symbolic key or sort key</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Type:** A = Alphanumeric, N = Numeric  
**Length:** F = Fixed, V = Variable

**Maximum record size including field separator characters = 35**
### 1.6 FIELD DESCRIPTION RECORD TYPE 1

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Field name</td>
<td>A</td>
<td>V</td>
<td>14</td>
<td>Must obey the rules for COBOL data-names and not begin with a Z.</td>
</tr>
<tr>
<td>2</td>
<td>Security level</td>
<td>N</td>
<td>V</td>
<td>2</td>
<td>1 = Highest security level. 1≤ Level ≤ Value specified in Field 4 of the Catalogue Control record.</td>
</tr>
<tr>
<td>3</td>
<td>Start position</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td>1≤ Start position ≤ Record size</td>
</tr>
<tr>
<td>4</td>
<td>Length of field</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td>1≤ Length ≤ 120 and Start position + Length ≤ Record size</td>
</tr>
<tr>
<td>5</td>
<td>Field type</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Alphanumeric, N = Numeric</td>
</tr>
<tr>
<td>6</td>
<td>Decimal point location</td>
<td>A</td>
<td>V</td>
<td>4</td>
<td>L for Left or R for Right followed by up to 3 unsigned digits.</td>
</tr>
<tr>
<td>7</td>
<td>Key field indicator</td>
<td>N</td>
<td>F</td>
<td>1</td>
<td>1≤ Key no. ≤ Value specified in Field 8 of File name record type 3. 1 = major key field 0 = non-key field</td>
</tr>
<tr>
<td>8</td>
<td>Sequence order</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Ascending, D = Descending Used only for key fields.</td>
</tr>
<tr>
<td>9</td>
<td>Domain indicator</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>Y = Yes, N = No. Used for all fields except the major key.</td>
</tr>
</tbody>
</table>

**Type:** A = Alphanumeric, N = Numeric  
**Length:** F = Fixed, V = Variable  

Maximum record size including field separator characters = 41
### 1.7 FIELD DESCRIPTION RECORD TYPE 2

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Description of field contents</td>
<td>A</td>
<td>V</td>
<td>70</td>
<td>The contents of this field is entirely at the user's discretion. A field separator character is not used to terminate this field</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric  
Length: V = Variable

Maximum record size = 70 characters
## 2. RELATIONDICT SUBFILE

### 2.1 RELATION DICTIONARY RECORD

The subfile is maintained in ascending Relation name sequence.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Relation name</td>
<td>A</td>
<td>V</td>
<td>30</td>
<td>Must obey the rules for COBOL data-names and not begin with a Z.</td>
</tr>
<tr>
<td>2</td>
<td>Password</td>
<td>A</td>
<td>F</td>
<td>8</td>
<td>Password for the highest security level of data to which the user may have access.</td>
</tr>
</tbody>
</table>

Type:  A = Alphanumeric, N = Numeric
Length:  F = Fixed, V = Variable

Maximum record size including field separator characters = 40
3. SUBMODEL SUBFILE

3.1 CONTROL RECORD

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No. of File descriptions in the Submodel</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td>This will be less than or equal to the number of File descriptions in the Catalogue.</td>
</tr>
<tr>
<td>2</td>
<td>No. of records in the Submodel subfile</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td>This will be less than or equal to the number of records in the Catalogue.</td>
</tr>
</tbody>
</table>

Type: N = Numeric
Length: V = Variable

Maximum record size including field separator characters = 12

3.2 FILE NAME RECORD TYPE 1

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File name</td>
<td>A</td>
<td>V</td>
<td>30</td>
<td>The SUBMODEL file descriptions are maintained in File name order.</td>
</tr>
<tr>
<td>2</td>
<td>No. of fields described in the Submodel</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td>This will be less than or equal to the no. of fields described in the Catalogue.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: V = Variable

Maximum record size including field separator characters = 36

N.B. There are no File name records of type 1C or 2 in the SUBMODEL subfile.
### 3.3 FILE NAME RECORD TYPE 3

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File medium</td>
<td>A</td>
<td>F</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Maximum block size</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Maximum record size</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>File label</td>
<td>A</td>
<td>F</td>
<td>12</td>
<td>This record is identical with its CATALOGUE subfile counterpart.</td>
</tr>
<tr>
<td>5</td>
<td>Access mode</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Organisation</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Length of symbolic key</td>
<td>N</td>
<td>V</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>No. of fields forming the symbolic or sort key</td>
<td>N</td>
<td>V</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 35
### 3.4 FIELD DESCRIPTION RECORD

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Field name</td>
<td>A</td>
<td>V</td>
<td>15</td>
<td>The Field name from the CATALOGUE subfile with the possible addition of a numeric suffix for non-key fields whose names are also in another file.</td>
</tr>
<tr>
<td>2</td>
<td>Start position</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Length of field</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Field type</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Decimal point location</td>
<td>A</td>
<td>V</td>
<td>4</td>
<td>These fields are identical to those in the corresponding Field description record type 1 in the CATALOGUE subfile.</td>
</tr>
<tr>
<td>6</td>
<td>Key field indicator</td>
<td>N</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Sequence order</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Domain indicator</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 39
### 4. DOMAINDICT SUBFILE

#### 4.1 CONTROL RECORD

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Domain count</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td>Count of Domain description records in the DOMAINDICT subfile. A field separator character is not used to terminate this field.</td>
</tr>
</tbody>
</table>

Type: N = Numeric  
Length: V = Variable

Maximum record size = 5 characters
### 4.2 DOMAIN DESCRIPTION RECORD

These records are maintained in ascending Domain name sequence.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Field name (Domain name)</td>
<td>A</td>
<td>V</td>
<td>15</td>
<td>The field name from the CATALOGUE subfile with the possible addition of a numeric suffix for non-key fields whose names are also in another file.</td>
</tr>
<tr>
<td>2</td>
<td>Length of field</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Field type</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Alphanumeric, N = Numeric</td>
</tr>
<tr>
<td>4</td>
<td>Decimal point location</td>
<td>A</td>
<td>V</td>
<td>4</td>
<td>L for Left or R for Right followed by up to 3 unsigned digits.</td>
</tr>
<tr>
<td>5</td>
<td>Key field indicator</td>
<td>N</td>
<td>F</td>
<td>1</td>
<td>0 = non-key field, 1 = major key field</td>
</tr>
<tr>
<td>6</td>
<td>Sequence order</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Ascending, D = Descending. Used only for key fields.</td>
</tr>
<tr>
<td>7</td>
<td>Domain indicator</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>Y = Yes, N = No. Used for all fields except the major key.</td>
</tr>
</tbody>
</table>

**Type:** A = Alphanumeric, N = Numeric  
**Length:** F = Fixed, V = Variable

Maximum record size including field separator characters = 34
5. **KEYDICT SUBFILE**

5.1 **KEY DICTIONARY RECORD**

This subfile always contains nine records not all of which may be in use. The records are in major to minor key order and the number of records currently in use is given by the value in global variable $^\#NKEY$.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Key domain name</td>
<td>A</td>
<td>V</td>
<td>14</td>
<td>This field is not terminated by a field separator character.</td>
</tr>
</tbody>
</table>

Type: $A =$ Alphanumeric  
Length: $V =$ Variable

Maximum record size = 14 characters

6. **DOMAINLIST SUBFILE**

6.1 **CONTROL RECORD**

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Domain count</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td>Count of Domain description records in the DOMAINLIST subfile. Field separator omitted.</td>
</tr>
</tbody>
</table>

Type: $N =$ Numeric  
Length: $V =$ Variable

Maximum record size = 5 characters
6.2 DOMAIN DESCRIPTION RECORD

These records are maintained in ascending Domain name sequence.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Field name</td>
<td>A</td>
<td>V</td>
<td>15</td>
<td>Domain name</td>
</tr>
<tr>
<td>2</td>
<td>Totalling marker</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>T = Totals required blank = Totals not required</td>
</tr>
<tr>
<td>3</td>
<td>Length of field</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Field type</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Alphanumeric N = Numeric</td>
</tr>
<tr>
<td>5</td>
<td>Decimal point</td>
<td>A</td>
<td>V</td>
<td>4</td>
<td>L for Left or R for Right followed by up to 3 unsigned digits. Used only for numeric fields</td>
</tr>
<tr>
<td>6</td>
<td>Key field indicator</td>
<td>N</td>
<td>F</td>
<td>1</td>
<td>0 = non-key field 1 = major key field</td>
</tr>
<tr>
<td>7</td>
<td>Sequence order</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Ascending D = Descending Used only for key fields.</td>
</tr>
<tr>
<td>8</td>
<td>Domain indicator</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>Y = Yes, N = No. Used for all fields except the major key.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 36
7. DOMAINDX INDEX SUBFILE

7.1 DOMAIND DESCRIPTION RECORD

This subfile is maintained in ascending Domain name sequence.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Domain or Item name</td>
<td>A</td>
<td>V</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Totalling marker</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>T = totals required blank = totals not required S = totals required for temporary item E = totals not required for temporary item</td>
</tr>
<tr>
<td>3</td>
<td>Length of field</td>
<td>N</td>
<td>V</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Field type</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Alphanumeric N = Numeric</td>
</tr>
<tr>
<td>5</td>
<td>Decimal point location</td>
<td>A</td>
<td>V</td>
<td>4</td>
<td>L for Left or R for Right followed by up to 3 unsigned digits.</td>
</tr>
<tr>
<td>6</td>
<td>Key field indicator</td>
<td>N</td>
<td>F</td>
<td>1</td>
<td>0 = non-key item 1 = major key item</td>
</tr>
<tr>
<td>7</td>
<td>Sequence order</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Ascending D = Descending Used only for key fields.</td>
</tr>
<tr>
<td>8</td>
<td>Domain indicator</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>Y = Yes, N = No. Used for all fields except the major key.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 36
8. LABELTABLE SUBFILE

8.1 LABEL RECORDS

The file contains 19 records and is maintained in alphabetical order of Label character. Fields 2, 3 and 4 are empty for unused Label characters.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Label character</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>The following are valid label characters: A E F G H I J K L N O Q T U V W X Y Z</td>
</tr>
<tr>
<td>2</td>
<td>Domain or Temporary item name</td>
<td>A</td>
<td>V</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Totalling marker</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>T = totals required blank = totals not required S = totals required for temporary item E = totals not required for temporary item</td>
</tr>
<tr>
<td>4</td>
<td>Field type</td>
<td>A</td>
<td>F</td>
<td>1</td>
<td>A = Alphanumeric N = Numeric</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric, N = Numeric
Length: F = Fixed, V = Variable

Maximum record size including field separator characters = 22
## 9. SUBFILES WHOSE NAMES HAVE THE PREFIX PAGE

### 9.1 CONTROL RECORD

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Line count</td>
<td>N</td>
<td>V</td>
<td>2</td>
<td>Count of the number of line records in the subfile. Field separator omitted.</td>
</tr>
</tbody>
</table>

Type: N = Numeric  
Length: V = Variable

Maximum record size = 2 characters

### 9.2 LINE RECORD

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Line format</td>
<td>A</td>
<td>V</td>
<td>70</td>
<td>Non-blank line formats are as entered by the user but with the visible space (#) and text delimiter (') characters replaced by blanks. Blank lines, specified explicitly or by default, are denoted by a greater than character (&gt;) in the first position. This is because the FREAD macro-time statement is unable to read blank, i.e. empty, subfile records. Field separator character omitted.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric  
Length: V = Variable

Maximum record size = 70 characters
10. PICTUREDICT SUBFILE

10.1 PICTURE DICTIONARY RECORD

This subfile is maintained in ascending Domain name order. The second field is not delimited by a comma separator character as this is a valid picture string character.

<table>
<thead>
<tr>
<th>Field No.</th>
<th>Contents</th>
<th>Type</th>
<th>Length</th>
<th>Max. size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Domain or Temporary item name</td>
<td>A</td>
<td>V</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>COBOL picture string</td>
<td>A</td>
<td>V</td>
<td>30</td>
<td>Field separator character omitted.</td>
</tr>
</tbody>
</table>

Type: A = Alphanumeric
Length: V = Variable

Maximum record size including the field separator character = 46
APPENDIX VI
LANGUAGE FACILITIES FOR COBOL GENERATION

1. BASIC STRUCTURE 1
2. DATA TYPES 1
3. INPUT/OUTPUT 2
4. PROGRAM CONTROL AND DATA MANIPULATION 2
5. ERROR DETECTION 3
APPENDIX VI

LANGUAGE FACILITIES FOR COBOL GENERATION

This appendix contains an initial attempt at defining the facilities of a high level language which are desirable for COBOL generation and is based on the experience gained while using the PG/2 macro processor. The material is presented under the following headings:

1. Basic structure
2. Data types
3. Input/Output
4. Program control and data manipulation
5. Error detection

1. BASIC STRUCTURE

1. Facilities for subroutines with names and arguments.
2. Overlay or chaining facilities.
3. Comments facility for annotating source code.
4. Access to assembly language subroutines, if necessary.

2. DATA TYPES

1. Integer and Real variables.
2. Variable length character strings.
3. Logical variables.
4. Global or Common variables.
5. Arrays with at least two dimensions.
6. Access to special operating system variables, e.g. date and time.
7. Use of literals - alphabetic, numeric and alphanumeric.
3. **INPUT/OUTPUT**

1. Efficient access to backing store files.
2. Sequential, random and/or indexed sequential access to records in backing store files.
3. Data editing facilities, e.g. insertion of currency symbols and punctuation characters etc.
4. Formatted and unformatted input/output.

4. **PROGRAM CONTROL AND DATA MANIPULATION**

1. Unconditional and conditional control statements.
2. Arithmetic expressions with operators for addition, subtraction, multiplication and division of variables.
3. Comparison operators for integer, real and character string variables, i.e. , , , and .
4. Logical operators AND and OR.
5. Numeric character conversion to real or integer and vice versa.
6. Variable and fixed length string extraction facilities.
7. Concatenation of strings.
8. String processing facilities for pattern matching.
9. Facilities for removing and/or replacing characters in strings.
10. Facilities for determining the length of a character string.
11. String class validation facilities, i.e. numeric, alphabetic or alphanumeric.
12. Macro expansion facilities for program generation (flexible templates).
5. ERROR DETECTION

1. Good diagnostics at compilation and execution time.
2. Error tracing facilities.
### APPENDIX VII

**COBOL REPORT GENERATING SYSTEM MACRO PROCESSING DETAILS**

<table>
<thead>
<tr>
<th>1. PROMPT ROUTINE</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. SEARCH ROUTINE</td>
<td>3</td>
</tr>
<tr>
<td>3. INVALID RESPONSE ROUTINE</td>
<td>3</td>
</tr>
<tr>
<td>4. DEFAULT RESPONSE ROUTINE</td>
<td>4</td>
</tr>
<tr>
<td>5. LIBPRELIM MACRO</td>
<td>4</td>
</tr>
<tr>
<td>6. LIBDELETE MACRO</td>
<td>6</td>
</tr>
<tr>
<td>7. LIBADD MACRO</td>
<td>7</td>
</tr>
<tr>
<td>8. STAGE3 MACRO</td>
<td>13</td>
</tr>
<tr>
<td>9. STAGE3-1 MACRO</td>
<td>16</td>
</tr>
<tr>
<td>10. STAGE4 MACRO</td>
<td>17</td>
</tr>
<tr>
<td>11. STAGE5 MACRO</td>
<td>19</td>
</tr>
<tr>
<td>12. STAGE5-1 MACRO</td>
<td>21</td>
</tr>
<tr>
<td>13. STAGE5-2 MACRO</td>
<td>24</td>
</tr>
<tr>
<td>14. STAGE6 MACRO</td>
<td>25</td>
</tr>
<tr>
<td>15. STAGE7 MACRO</td>
<td>27</td>
</tr>
<tr>
<td>16. STAGE8 MACRO</td>
<td>33</td>
</tr>
<tr>
<td>17. STAGE9 MACRO</td>
<td>35</td>
</tr>
<tr>
<td>18. IDENTIFICATION, VALIDATION AND GENERATION OF FIELD DESCRIPTIONS</td>
<td>41</td>
</tr>
<tr>
<td>19. WRITEPARA MACRO</td>
<td>48</td>
</tr>
<tr>
<td>20. TITLEPARA MACRO</td>
<td>49</td>
</tr>
<tr>
<td>21. PHEADPARA MACRO</td>
<td>50</td>
</tr>
<tr>
<td>22. SBHPARAn MACROS</td>
<td>51</td>
</tr>
<tr>
<td>23. STLPARAn MACROS</td>
<td>52</td>
</tr>
<tr>
<td>24. TOTALPARA MACRO</td>
<td>53</td>
</tr>
<tr>
<td>25. COMBINATIONS OF REPORT OUTPUT CATEGORIES AND ASSOCIATED PAGE SIZE CONDITIONS</td>
<td>55</td>
</tr>
<tr>
<td>26. OWName MACROS</td>
<td>57</td>
</tr>
<tr>
<td>27. OWN CODE STATEMENTS</td>
<td>59</td>
</tr>
</tbody>
</table>
Table 5.1 COBOL rules for data-names
Table 7.1 Response validation for File name records data
Table 7.2 Response validation for Field description record type 1 data
Table 12.1 Rules for a COBOL numeric literal
Table 12.2 Picture strings for numeric items
Table 15.1 Syntax validation for a Simple Condition located in the input buffer
Table 15.2 Validate 1st Operand as a domain or temporary item name
Table 15.3 Validate Condition Operator
Table 15.4 Identify nature of Second Operand
Table 15.5 Validation of Alphanumeric literal 2nd Operand
Table 15.6 Validation of Numeric literal 2nd Operand
Table 15.7 Validation of Domain or Temporary item as 2nd Operand
Table 17.1 Validate Alphanumeric editing format
Table 17.2 Character transfers to the output buffer during an Alphanumeric Edit
Table 17.3 Validate Numeric editing format
Table 17.4 Character transfers to the output buffer during a Numeric Edit - Part 1
Table 17.5 Character transfers to the output buffer during a Numeric Edit - Part 2
Table 17.6 Adjustments to the contents of the output buffer prior to printing edited numeric data
Table 18.1 Identification and validation of Blank FILLER field
Table 18.2 Identification and validation of Alphanumeric FILLER field
Table 18.3 Identification and validation of Alphanumeric data field
Table 18.4 Derivation of a picture string from an alphanumeric field format specification
Table 18.5  Identification and validation of Numeric data field  
Table 18.6  Derivation of a picture string from a numeric field format specification - Part 1  
Table 18.7  Derivation of a picture string from a numeric field format specification - Part 2  
Table 25.1  Page size conditions  
Table 26.1  Macro definition subfile names used for own code processing  
Table 27.1  Identification of the ADD statement and delimiters of operands  
Table 27.2  Permitted operand types for ADD, SUBTRACT and MULTIPLY  
Table 27.3  Operand replacement details for Arithmetic operations  
Table 27.4  Identification of the SUBTRACT statement and delimiters of operands  
Table 27.5  Identification of the MULTIPLY statement and delimiters of operands  
Table 27.6  Identification of the DIVIDE statement and delimiters of operands  
Table 27.7  Permitted operand types for DIVIDE  
Table 27.8  Identification of the MOVE statement and delimiters of operands  
Table 27.9  Permitted operand types for MOVE  
Table 27.10  Operand replacement details for MOVE operations  
Table 28.1  String and global variables in the CONSTANTS macro  
Table 30.1  Subfile organisation  
Table 30.2  Initialisation of 'grown' macro definition subfiles  
Table 30.3  Initialisation of common data subfiles
APPENDIX VII

COBOL REPORT GENERATING SYSTEM MACRO PROCESSING DETAILS

The material in this appendix supplements Chapter 7 of the main text by providing further details about the following:

1. Frequently used routines.
2. Validation of user responses.
4. Problem specifying macros.
5. 'Grown' macro outlines.

All macros used during the problem specifying dialogue have one parameter which is always the % character. When macro definitions are 'grown' the parameter is used in the generation of system macro and macro-time statements (c.f. Section 3.6.2 of the main text). A change in the operating system used for the development of the COBOL generating system enabled @, the default parameter marker character, to be used (Appendix I Sections 3 and 6.1).
1. PROMPT ROUTINE

This routine issues a prompt character, reads the user's reply and detects a plea for help. The routine is present in all the problem specifying macros which require the user to enter a response.

The routine forces the operating system invitation to type character (←) on to a new line by writing out a colon character thus:

:\←

The user then has up to 70 characters available for typing his response line. The response is read, the first five characters are extracted from the input buffer and tested to see if they are '%HELP'. If help has been requested the routine chains to the HELP macro, otherwise control returns to the statement following that which called the routine. The statement to which control is returned is usually the first of several which validate the response.

The listing of the STAGE2 macro (Item 18 in the separate folder) illustrates the use of this routine, the first statement of which is labelled 9000. In macros not in the same mainfile as the HELP macro, the CONTINUE statement with the label 9001 is replaced by the following:

%9001 MAINFILE,GWAC132-DATA
2. SEARCH ROUTINE

This routine carries out a binary search of the Domain dictionary looking for a particular domain or temporary item name.

The Domain dictionary subfile, DOMAININDEX, is held in alphabetical order of domain name, which occupies the first field of each record (Appendix V Section 7). The count of the number of records in the subfile is maintained in global variable $GDMCT.

The routine expects the name of the domain for which the search is being made to be stored in string variable $S09. The routine sets local variable $IND equal to zero if the search is successful and equal to 1 if it is not.

A DOMAININDEX record is read and stored in string variable $S01, from which the domain name field is extracted into string variable $S02. Local variables $LOW, $HIGH and $CUR are used to calculate the number of the record to be read.

If a search is unsuccessful, local variable $LOW points to the position in the subfile where, if desired, a new record containing details about the $S09 domain may be inserted.

The listing of the STAGE5 macro (Item 22 in the separate folder) illustrates the use of this routine, which begins with the statement labelled 8000.

3. INVALID RESPONSE ROUTINE

This routine outputs the 'INVALID RESPONSE' message.

All user responses are validated and where possible a meaningful error message is output before the invitation to type is reissued. There are, however, many instances where a simple response of one or two characters is required. Here an error, if it occurs, is usually obvious and often only a typing error. In
such circumstances use is made of this routine which merely prints
the words 'INVALID RESPONSE'. On return from this routine control
is passed again to the prompt issuing statement, the label of
which appears as the second parameter of the OBEY macro-time
statement which initiated the call to the routine.

The listing of the STAGE5 macro (Item 22 in the separate
folder) illustrates the use of this routine, which begins with
the statement labelled 2100.

4. DEFAULT RESPONSE ROUTINE

This routine outputs the 'Default response' message.

During the course of the problem specifying dialogue there
are many instances where a yes/no answer is required to a question.
This routine, which instructs the user to enter his reply as a
single character 'Y' or 'N' or to accept 'no' as the default reply,
is called for this purpose.

The routine is illustrated in the listing of the STAGE5
macro (Item 22 in the separate folder) and begins at the statement
labelled 2200.

5. LIBPRELIM MACRO

Further details of the five processing tasks of the LIBPRELIM
macro are given below. On the listing of the macro (Item 14 in
the separate folder) the beginning of each task has been
highlighted.

1. The user is prompted to enter one of three valid
options - ADD, DELETE or FINISH. Any other response by the user
causes an error message to be output and the prompt is reissued.

2. The dialogue prompts the user to enter the name of the
file whose description is to be added or deleted. The file name
is validated to ensure that it conforms with the COBOL rules for data-names (Table 5.1). An additional restriction, i.e., data-names may not begin with a Z, is imposed to avoid user file names duplicating those generated in the COBOL program which all have Z as a prefix. The user is only reminded of the rules for the formation of data-names when he infinges one of them. He is then prompted to enter a different name.

3. The user is prompted to enter the top security level password (level 1) for the file.

4. The Filename type 1 records of the CATALOGUE subfile are searched to see if the user's file name is already in the Catalogue. The count of fields described in the current file description is extracted and used together with the number of security levels used to protect the Catalogue to calculate the number of the next record to be examined. As the Catalogue file descriptions are recorded in ascending file name order, the search stops when one of the following three conditions is satisfied:

- The required file name is found.
- All the Filename type 1 records in the CATALOGUE subfile have been examined without finding the required file name.
- A file name greater than that entered by the user is found.

Global variable $GERR is set equal to zero or 1 depending on whether or not the required file name is found. If the file name is located but the user has not supplied the correct level 1 password, global variable $GERR is set equal to 1, otherwise it is set equal to zero. These global variables are interrogated in the LIBADD and LIBDELETE macros.

5. Control passes to the LIBADD or LIBDELETE macro depending on the option selected at the beginning of the LIBPRELIM macro. Global variable $GOPT contains the option chosen.
Table 5.1 COBOL rules for data-names

<table>
<thead>
<tr>
<th>Not more than 30 characters long</th>
</tr>
</thead>
<tbody>
<tr>
<td>No imbedded blanks</td>
</tr>
<tr>
<td>Only alphabetic, numeric and hyphen characters allowed</td>
</tr>
<tr>
<td>First and last characters may not be hyphens</td>
</tr>
<tr>
<td>At least one alphabetic character must be present</td>
</tr>
</tbody>
</table>

6. LIBDELETE MACRO

Further details of the processing tasks carried out by the LIBDELETE macro are given below. On the macro listing (Item 15 in the separate folder) the beginning of each task has been highlighted.

1. The Catalogue search carried out by the LIBPRELIM macro must have been successful, i.e. the #FERR global variable used as the file name error indicator must contain zero. If a non-zero value is detected this indicates that the given file is not described in the Catalogue and a message to this effect is output on the user's terminal.

2. The user must have been able to supply the top security password (level 1) for the file in the LIBPRELIM macro, i.e. global variable #PERR must contain zero. If a non-zero value is detected the user is advised that his password is unacceptable and that his request to delete the file description has been cancelled.

3. After the successful search of the CATALOGUE subfile in the LIBPRELIM macro, the record number global variable, #REC, points at the first record of the file description to be deleted. The number of records to be deleted is a function of the number of fields described in the file and the number of security levels by which the Catalogue data is protected. The DELETE macro-time
statement is executed the required number of times by means of a program loop.

4. The file and record counts from the Catalogue Control record, which were stored in global variables $\#GFCT$ and $\#GRCT$ during the LIBRARIAN macro, are reduced appropriately. Their new values together with the Catalogue password, number of security levels and the current date and time are concatenated into a string variable which is used to replace the CATALOGUE subfile Control record.

5. The user is advised of the successful deletion by a message on his terminal.

6. No matter whether or not the request to delete a file description was successfully effected control is always returned to the LIBPRELIM macro, which permits processing to continue.

7. LIBADD MACRO

The processing tasks carried out by the LIBADD macro are given in further detail below. The beginning of each task has been highlighted on the macro listing (Item 16 of the separate folder).

1. The Catalogue search carried out by the LIBPRELIM macro must have been unsuccessful, i.e. the $\#GFERR$ global variable used as the file name error indicator must be non-zero. A zero value indicates that the file with the given name is already in the CATALOGUE subfile and a message to this effect is output on the user's terminal.

2. On entry to the LIBADD macro $\#GREC$, the global variable used as the record pointer, points to the position where the new file description should be inserted. Throught the processing of the LIBADD macro this variable is incremented by 1 each time a record is inserted in the CATALOGUE subfile.
3. The file name and highest security level password for the file whose description is being added to the Catalogue were gathered during the LIBPRELIM macro. It therefore remains to request the user to supply the remaining data for the other fields of the File name records. The data for each field is suitably validated and concatenated into a string variable with the fields separated by commas. As soon as a file name record is complete it is entered into the CATALOGUE subfile, thus releasing the string variable for other use. As not all the fields in the File name records may apply to a particular file the dialogue is programmed so that only pertinent questions are asked. The comma separator character only is concatenated into the record to indicate an unused field. Although the dialogue gives the user some guidance about what constitutes a valid response, extensive validation of the user’s replies is vital for the integrity of the Catalogue data. The nature and extent of the reply validation for the File name records data is given in Table 7.1 and their formats are shown in Section 1 of Appendix V.

4. As the number of fields in a record of the file currently being described was entered as data for the File name type 1 record, a processing loop is set up to gather data for each pair of Field description records. The data to be gathered about each field depends on its type, numeric or alphanumeric, and whether or not it is a sequence key field. The dialogue is again programmed to pose only relevant questions and for unused fields only the comma separator character is concatenated into the record description. The dialogue prompts give the user guidance on the form of each reply which is then suitably validated. Details of the validation for Field description type 1 record data are given in Table 7.2. The type 2 Field description record is purely descriptive and requires no validation.
As each key field description is processed the key number is concatenated into string variable #S06.

5. When all the pairs of valid Field description records have been inserted into the CATALOGUE subfile it is necessary to check that all key fields have been allocated. The string variable #S06 contains the concatenated key numbers, which are single digits separated by commas. #S06 is inspected to see that it contains all the key numbers for the file. If any key number is found to be missing the whole file description is deleted from the Catalogue and an error message is output on the user's terminal.

6. The file and record counts from the Catalogue Control record, which were stored in global variables #GFCT and #GRCT during the LIBRARIAN macro, are incremented appropriately. Their new values together with the Catalogue password, number of security levels and the current date and time are concatenated into a string variable which is used to replace the CATALOGUE subfile Control record.

7. The user is advised by a message on his terminal that the file description has been successfully inserted in the CATALOGUE subfile.

8. No matter whether or not the request to add a file description was successful control is always returned to the LIBPRELIM macro which permits processing to continue.

In task 4 above the user is encouraged to enter field descriptions in ascending start position order as this leads to more efficient processing. The savings are made later when the Catalogue data is used to generate a COBOL record description for the file. As the PG/2 macro processor has no sorting facility it is not possible to rearrange the Catalogue field description
records into start position order at a later date. At COBOL generation time a REDEFINES statement has to be generated whenever the start position of a field is not greater than the end position of the previous field described in the Catalogue (Appendix VIII Section 4).

The method of handling an error in task 5 above may seem rather drastic, but it is adopted because the LIBADD macro is already near to the size limit imposed by the PG/2 macro processor. Additional recovery processing would necessitate the subdivision of the LIBADD macro into several chained macros. While the restriction is reasonable in an 'experimental' implementation it does not prejudice the implementation of a more refined recovery procedure at a later date. The aim has been to provide the data base administrator with a simple tool for maintaining the Catalogue of file descriptions.
<table>
<thead>
<tr>
<th>Record Type</th>
<th>Field contents</th>
<th>Conditions to be satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of fields in a record</td>
<td>String of up to four digits with value ( \geq 1 )</td>
</tr>
<tr>
<td>2</td>
<td>Field description</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>File medium</td>
<td>2 character string with value: 'PT', 'CR', 'MT' or 'ED'.</td>
</tr>
<tr>
<td></td>
<td>Maximum block size</td>
<td>String of digits with value ( \geq 1 ) and ( \leq 2048 )</td>
</tr>
<tr>
<td></td>
<td>Record size</td>
<td>String of digits with value ( \geq 1 ) and ( \leq ) Maximum block size.</td>
</tr>
<tr>
<td></td>
<td>File label</td>
<td>12 character string for ED and MT files only.</td>
</tr>
<tr>
<td></td>
<td>Access mode</td>
<td>Single character with value 'R' or 'S' for ED files only.</td>
</tr>
<tr>
<td></td>
<td>Organisation</td>
<td>Single character with value 'I' or 'D' for ED random access files only.</td>
</tr>
<tr>
<td></td>
<td>Length of symbolic key</td>
<td>String of digits with value ( \geq 1 ) and ( \leq 64 ) for ED random access files only.</td>
</tr>
<tr>
<td></td>
<td>Number of fields in symbolic or sort key</td>
<td>Single digit with value ( \geq 0 )</td>
</tr>
</tbody>
</table>
Table 7.2 Response validation for Field description record type 1

<table>
<thead>
<tr>
<th>Field contents</th>
<th>Conditions to be satisfied by user's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field name</td>
<td>Character string with length ≤14. No imbedded blank characters. First character not Z or hyphen. Last character not hyphen. Consist only of alphabetic, numeric and hyphen characters. Contain at least one alphabetic character. Must not duplicate the name of another field already in the same File description.</td>
</tr>
<tr>
<td>Security level</td>
<td>A character string which must not contain a comma, but which must be a substring of that in string variable #S08. (String variable #S08 contains all the valid security level values separated by commas and was set up in the LIBRARIAN macro.)</td>
</tr>
<tr>
<td>Start position</td>
<td>String of digits with value ≥1 and ≤record size.</td>
</tr>
<tr>
<td>Length of field</td>
<td>String of digits with value ≥1 and ≤120. Start position + Length of field - 1 ≤ record size.</td>
</tr>
<tr>
<td>Field type</td>
<td>A single character; either 'A' or 'N'.</td>
</tr>
<tr>
<td>Decimal point location</td>
<td>Character string with the first character either 'L' or 'R'. Subsequent characters to be digits with value ≤99. (Only applicable to numeric fields.)</td>
</tr>
<tr>
<td>Key field indicator</td>
<td>A single character which must be present in string variable #S07. (String variable contains 0 and all the valid key number digits for the file.) If non-zero the character must not be present in string variable #S06. (String variable #S06 contains all the previously allocated key numbers for the current file.)</td>
</tr>
<tr>
<td>Sequence order</td>
<td>A single character; either 'A' or 'D'. (Only applicable to key fields.)</td>
</tr>
<tr>
<td>Domain indicator</td>
<td>A single character; either 'Y' or 'N'. (Applicable to all fields except the major key.)</td>
</tr>
</tbody>
</table>
8. STAGE3 MACRO

Thirteen main processing steps are carried out during the execution of the STAGE3 macro and their details are as follows. The first statement of each step is highlighted on the macro listing (Item 19 in the separate folder).

1. Set the global variable used to contain the check point number ($GCKPT$) equal to 3.

2. Print out the Stage 3 introductory dialogue which is stored in the DIAL3 subfile. This text advises the user that he is about to be shown details of the domains within the relations of his data base. It goes on to explain with illustrations the notation used to indicate the position of the decimal point in a numeric field.

3. Establish details of the Catalogue characteristics by reading the data in the Catalogue Control record.

4. Carry out the housekeeping operations necessary to maintain pointers to the current record in the subfiles used by the macro, keep count of the number of files described in the SUBMODEL subfile, and count the number of fields described in the current SUBMODEL file description.

5. For each relation named in the RELATIONDICT subfile the CATALOGUE subfile is searched to see if there is a matching file description. Both subfiles are maintained in relation name (file name) sequence. When no matching name is found the error indicator is set, the user is advised that the relation is not in the data base and the search for the next relation in the Relation dictionary is begun. When a relation has a matching description in the Catalogue its name and the column headings for the domain details are printed.

6. The user supplied password from the RELATIONDICT record
is matched against the passwords in the Catalogue File description type 1 record to determine which security levels the user may access. If the Catalogue has more than four security levels it may be necessary to search the passwords in the type 1C record to obtain a match. When no matching password can be found the error indicator is set, the user is advised that as the password is invalid no domains are accessible and the search for the next relation in the Relation dictionary begins.

7. The Catalogue File name type 3 record is read and it is copied on to the SUBMODEL subfile after the shortened form of the type 1 record.

8. The Field description type 1 records of the current Catalogue file description are read in turn and checked to see if the user has password access to the field (domain). The security level of the field must be greater than or equal to the level set by the user's password in step 6 above.

9. For each Field description record to which the user is allowed access, the remaining fields are extracted into local variables ready for later use. The DOMAINDICT subfile, which is maintained in domain name sequence, is searched to see if the current field name (domain name) is already in the Domain dictionary. If it duplicates a dictionary entry and is not a key field then the ambiguity is resolved by appending the relation count as a suffix to the field name. The field name is padded out with relation count digits until it is 15 characters long. The details of the renamed field are inserted in the DOMAINDICT subfile. If the field name is not found in the Domain dictionary then its details are inserted in the DOMAINDICT subfile. Prior to that, if it is a key field, its name would have been entered in the KEYDICT subfile. This latter subfile is maintained in key number sequence and global variable $NKEY contains the subfile record
10. The current Catalogue Field description type 1 record to which the user has password access is copied to the SUBMODEL subfile. The details of name, size, type, and decimal point location, if applicable, are output to the user's terminal.

11. When all the field description records in the current Catalogue File description have been processed, the field count in the current Submodel File description record type 1 is updated. Only if the user has password access to all the fields in the file description will this be the same as it was in the Catalogue.

12. The domain names in the KEYDICT subfile for the current relation are output on the user's terminal in major to minor key order.

13. When all the relations in the KEYDICT subfile have been processed without error, the control records for the SUBMODEL and DOMAINDICT subfiles are written and control passes to the STAGE3-1 macro by means of the chain facility. If, however, the error indicator has been set a message is output to the user's terminal advising him that the error prevents further progress. The user is then given the choice of abandoning the run or returning to the STAGE2 macro to respecify the relation names and passwords.

The error handling facilities in this macro show potential for enhancement. A useful extension would be to list on the user's terminal the relation names and passwords entered in Stage 2 under three categories:

1. Invalid relation names.
2. Valid relation names with invalid passwords.
3. Valid relation names and passwords.

Facilities could then be provided so that only corrections to entries in the first two categories need be made before resuming the processing.
9. STAGE3-1 MACRO

This macro checks that the relations named by the user in Stage 2 are not disjointed or disconnected according to the criteria outlined in Section 6.4.2 of the main text. If only one relation was named in Stage 2 then this macro immediately chains to the STAGE4 macro. The checking process is carried out in six steps described below. On the macro listing (Item 20 in the separate folder) the statement at which each step begins has been highlighted.

1. The SUBMODEL subfile Control record is read in order to establish the number of file descriptions and records contained therein.

2. Local variables are set up for use as pointers and counters to keep track of the current record in the subfiles used by the macro. Global variable \#GNKEY is initially set to zero, ultimately it will contain the number of key domains in the joined relations.

3. A pass is made through the SUBMODEL subfile to extract and examine the pairs of File description records in order to establish which of the described files has the greatest number of key fields. When the number of keys in the current file description exceeds the value in \#GNKEY its value is updated and the position of the file description within the SUBMODEL subfile is noted. The field description records for this file are examined and the key field names are extracted and stored in the KEYDICT subfile in major to minor key order.

4. A second pass through the SUBMODEL subfile is made in order to compare the key details all file descriptions, other than the one noted during pass one, with the contents of the KEYDICT subfile.
5. If the key field names of any of the files do not form an ordered contiguous subset of those in the Key dictionary, the relations are deemed to be disjointed or disconnected. If this condition is detected a message is output on the user's terminal telling him that his relations cannot be joined. He is then given the opportunity to return to Stage 2 to respecify the relation names and passwords or to abandon the run.

6. If no errors are detected during the processing of this macro control chains to the STAGE4 macro.

At present little is done in the STAGE3-1 macro to help the user if the relations he specifies are found to be disjointed or disconnected. He can only seek advice from the data base administrator. Future enhancements could include the output of a list of those relations which cannot be joined, together with details of the key domains which cause them to be disjointed or disconnected. Only details of these latter relations need then be changed before processing resumes with the STAGE2 macro.

10. STAGE4 MACRO

The processing carried out by the STAGE4 macro falls into twelve main steps which are described below. On the macro listing (Item 21 in the separate folder) the statement at which each step begins has been highlighted.

1. Set the check point variable CKPT equal to 4, ready in case the restart facility is invoked by a call for 'help' by the user.

2. Output the introductory text for this stage which is stored in the DIAL4 subfile.

3. Set up and initialise local variables for use as record pointers to the DOMAINDICT and DOMAINLIST subfiles.
4. Create and set to zero the global variable $TOT which is used to indicate whether or not the totalling facility is required.

5. Select the DOMAINDICT subfile, read the Control record, extract and convert to binary the count of domain records stored in the subfile.

6. Read the domain description from the DOMAINDICT subfile, record by record, and extract the fields for processing.

7. The name of each domain in turn is displayed to the user who is asked to indicate, by a yes or no reply, whether it is required to solve the problem.

8. For each numeric domain which features in the problem the user is additionally asked if the totalling facility is required for it. Again a yes/no answer is called for.

9. When the totalling facility is invoked for any numeric domain the global variable indicator $TOT is set equal to 1.

10. Of the domains which are required to solve the problem, the alphanumeric and non-totalling numeric ones have their details written to the DOMAINLIST subfile with the totalling marker set to a blank character. Numeric domains for which totalling is required have the totalling marker set equal to the character 'T'. In addition details of key domains not required to solve the user's problem are by default entered in the DOMAINList subfile with a blank totalling marker. Their details are required when the complete COBOL program is generated.

11. A count of domain records written to the DOMAINLIST subfile is maintained. When all the domains in the DOMAINDICT subfile have been processed, the value of this count is stored in the Control record of the DOMAINLIST subfile.

12. The final task is to chain to the STAGE5 macro.
11. STAGE5 MACRO

The processing within this macro consists of the steps shown below. The first statement of each step is highlighted on the macro listing (Item 22 in the separate folder).

1. Set the global check point variable $GCKPT equal to 5, ready for restart procedures in invoked by the user's request for help.

2. Print the introductory dialogue which describes the facilities available in Stage 5. The text of this dialogue is stored in subfile DIAL5.

3. Select the DOMAINLIST subfile, read the Control record and convert the domain count into binary. This count is stored in global variable $GDMCT and from it the number of the last record in the DOMAINLIST subfile may be calculated.

4. Set up and maintain record pointers for both the DOMAINLIST and DOMAININDEX subfiles.

5. Copy all the domain description records in the DOMAINLIST subfile to DOMAININDEX.

6. Generate the first two statements of the TEMPITEM macro definition.

7. Ask if the user wishes to create any temporary items. If the reply is 'no' then control passes to the STAGE5-2 macro by means of the chain facility. Otherwise, processing continues as outlined below.

8. The user wishes to create temporary items, so he is now asked if any of them contain alphanumeric data. If the reply is in the negative the chain facility passes control to the STAGE5-1 macro. Otherwise, details of the alphanumeric items are gathered in the manner indicated below.

9. Request the name of the new alphanumeric item and validate the user's reply. The name must satisfy the COBOL rules
for a dats-name (Table 5.1 of this appendix) and two additional conditions imposed by the COBOL generating system. First, that the name must not exceed 14 characters in length and secondly, that it must not begin with the character 'Z'.

10. The user entered name must be unique, so a binary search of the names in the records of the DOMAININDEX subfile must be carried out in order to establish this. Steps 9 and 10 are repeated until the user enters a unique valid name.

11. The user is asked to state how many characters the new alphanumeric item contains. The reply must be a number in the range 1 to 58 with a default value of 30. The upper limit is less than that permitted by COBOL and the reason for the restriction is explained in step 14.

12. The initial value of the alphanumeric item may be set by the user, so he is asked if he wishes to avail himself of the facility. If the reply is 'yes' then the user is prompted to enter the value. Otherwise, the initial value is set, by default, to contain blanks.

13. All the information necessary to set up and insert a description record for the new item in the DOMAININDEX subfile is to hand. The insertion position for the record is stored in local variable $LOW as a by-product of the binary search carried out in step 10 above. The domain count variable $GDMCT is also incremented by 1.

14. Finally, the data description entry for the new alphanumeric item is generated. In order to avoid the special coding necessary when a COBOL literal continues onto another line, the maximum size of a temporary item is limited to 58 characters. This is the largest literal, including quotation mark delimiters and the final full stop character, which can be
accommodated between positions 12 and 72 of a COBOL statement. The generated COBOL statements have the following form:

02 temporary-item-name PICTURE X(item-size) VALUE "initial-value".

15. Steps 9 to 14 are repeated until the user gives a negative reply when asked if he has any more alphanumeric items to create. Control then passes to the STAGE5-1 macro by means of the chain facility.

12. STAGE5-1 MACRO

In this macro for creating temporary numeric domains, the processing is carried out as shown below. On the macro listing (Item 23 in the separate folder) the first statement of each step has been highlighted.

1. Ask the user if he wishes to create any temporary numeric items. If the reply is 'no', then control passes to the STAGE5-2 macro by means of the chain facility. Otherwise, the processing continues as outlined below.

2. The name of the numeric item is requested and validated in a similar way to that described in steps 9 and 10 of the STAGE5 macro.

3. The user is asked to state how many characters the numeric item contains. The reply must be a number in the range 1 to 60, with a default value of 18. The upper limit is less than that permitted by COBOL, but it is the largest size which, together with the full stop character, can be accommodated in one line of COBOL coding (c.f. STAGE5 step 14).

4. The location of the decimal point within the numeric item is established by asking the user to indicate its position relative to the right-most digit. The direction is specified by
a single character, 'L' for left or 'R' for right. The displacement must be entered as a single digit, i.e. in the range 0 to 9, with a default value of zero. This is an arbitrary restriction which could be relaxed at a later date should circumstances require.

5. The user is asked if he wishes to enter an initial value for the temporary numeric item. The default reply is 'no', in which case the item is initially set to zero. If the user replies 'yes' he is prompted to enter the initial value. The response is validated to ensure that it obeys the rules for a COBOL numeric literal (Table 12.1 of this appendix).

6. The automatic totalling option is available for temporary numeric items, so the user is invited to state if he wishes to avail himself of the facility. The default reply is 'no'. If totalling is required, the global variable totalling indicator, #GTOT, is set equal to 1 and the local variable used as the totalling marker for the item is set equal to 'S'. If totalling is not required, the totalling marker is set equal to 'E'.

7. All the information necessary to set up and insert a description record for the new temporary numeric item in the DOMAININDEX subfile is to hand. The record insertion position was set during step 2 above (c.f. STAGE5 macro). The domain count in global variable #GDMCT is incremented by 1.

8. The COBOL data description entry for the new numeric item can now be generated and takes the following form:

```
02 temporary-item-name PICTURE picture-string COMPUTATIONAL VALUE initial-value.
```

Depending on the item size and the location of the decimal point, one of five picture strings is generated. These details are summarised in Table 12.2 of this appendix.

9. Steps 2 to 8 are repeated until the user gives a negative
reply when asked if he has any more numeric items to create. The processing in the latter part of step 1 is then carried out.

Table 12.1 Rules for a COBOL numeric literal

<table>
<thead>
<tr>
<th>Decimal point location</th>
<th>Picture-string</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Within the string of digits.</td>
<td>$S9(n)V9(m)$</td>
</tr>
<tr>
<td>2. Immediately after the last digit.</td>
<td>$S9(1)$</td>
</tr>
<tr>
<td>3. One or more positions to the right of the last digit.</td>
<td>$S9(1)P(m)V$</td>
</tr>
<tr>
<td>4. One or more positions to the left of the first digit.</td>
<td>$SVP(p)9(1)$</td>
</tr>
<tr>
<td>5. Immediately before the first digit.</td>
<td>$SV9(1)$</td>
</tr>
</tbody>
</table>

where $l = \text{item size}$, $m = \text{decimal point shift count}$, $n = l - m$, and $p = m - l$
13. STAGE5-2 MACRO

The processing in the STAGE5-2 macro is carried out as shown below. On the macro listing (Item 24 in the separate folder) the first statement of each step has been highlighted.

1. The statements for the TEMPITEM macro generated during the execution of macros STAGE5 and STAGE5-1 are saved in the TEMPITEM subfile. The first two statements for the CONCAT macro are generated and saved in the CONCAT subfile.

2. The remaining text in the DIAL5 subfile, which describes the concatenation of domains, is output and the user is asked if he wishes to create any compound domains. If the user replies 'yes', then processing continues at step 3. Otherwise, the final statements for the CONCAT and TEMPITEM macros are generated and filed and control chains to the STAGE6 macro.

3. The name of the compound domain is requested in a similar way to that described in steps 9 and 10 of the STAGE5 macro.

4. A count of the number of characters in the compound domain is maintained in local variable $LCHCT and this is initially set equal to zero.

5. A copy of the binary search record pointer, used in step 3 above, is saved in a local variable. It is used later in step 11 when the compound domain description is inserted in the DOMAININDEX subfile.

6. The COBOL level 02 data description statement for the group item is generated and appended to the TEMPITEM subfile. The generated statement has the following form:

   O2 compound-domain-name.

7. The user is requested to enter the name of a domain to be concatenated. A binary search of the names in the DOMAININDEX subfile is then made to check that the domain exists within the user's data base. If the domain name cannot be found the user is
advised and another name must be entered.

8. The size of items in the domain to be concatenated is extracted from its DOMAININDEX record, converted to binary and stored in a local variable. This value is added to the variable created in step 4 above for maintaining the compound domain size.

9. The COBOL level 03 data description for the domain being concatenated to form the compound domain is generated and appended to the TEMPITEM subfile. The statement generated has the form:

   03 domain-name PICTURE X(item-size).

10. The Procedure division MOVE statement which transfers data from the individual item to the corresponding sub-item of the group is generated and appended to the CONCAT subfile. The generated statement occupies two lines and has the following form:

   MOVE domain-name IN Z-TUPLE TO domain-name IN compound-domain-name IN Z-TUPLE.

11. Steps 7 to 10 are repeated until the user has no more domains to include in the current compound domain. The compound domain description record is then set up and inserted in the DOMAININDEX subfile at the position determined in step 5 above. Also the domain count in #GDMCT is incremented by 1.

12. For the information of the user the size of the newly created compound domain is displayed on the user's terminal.

13. Steps 3 to 12 are repeated until the user gives a negative reply when asked if he has any more compound domains to create. The processing described in the latter part of step 2 is then carried out.

14. STAGE6 MACRO

The following steps give details of the STAGE6 macro processing. On the macro listing (Item 25 in the separate folder) the first statement of each step has been highlighted.
1. The check point variable $GCKPT$ is set equal to 6 ready in case the 'help' facility is invoked.

2. If the number of relations ($GREL$) required to solve the user's problem is equal to 1, control passes to step 9. Otherwise, processing continues below.

3. Part of the descriptive text stored in the DIAL6 subfile is printed on the user's terminal. This describes the options open to the user when a data inconsistency occurs during the execution of the COBOL program. The user is then prompted to select an option.

4. The user's response is validated and if the default option was selected the value is specifically set equal to 1. The option number is converted to binary and stored in global variable $GOPT$ for use in a later stage.

5. The user is asked to indicate if he wishes to have a message printed by the COBOL program when a mismatch condition is detected in the data files. The user's reply is validated and stored in global variable $GMOP$. If the reply is explicitly or by default 'no' control passes to step 9, otherwise processing continues below.

6. The remaining text in the DIAL6 subfile is printed. This describes the form and content of the error message and asks the user if he wishes to specify some text of his own for inclusion in the message.

7. The user's response to the above question is validated and interpreted. When the default message is selected this is set up and stored in string variable $S05$ and control passes to step 9, otherwise processing continues below.

8. As the user has opted to include his own text in the program error message, he is prompted to enter up to 20 characters. Like the default message these characters are stored in string
variable #S05.

9. The chain facility passes control to the STAGE7 macro.

15. STAGE7 MACRO

The processing in the STAGE7 macro is as set out below. On the macro listing (Item 26 in the separate folder) the first statement of each step has been highlighted.

1. Set the check point variable #CKPT equal to 7.

2. Generate the first two statements of the SELCOND macro which will contain the COBOL selection condition statements, if any, for data retrieval.

3. Ask the user if he wishes to select only certain items for retrieval from his domains. If the reply is 'no', then processing continues at step 7 below.

4. As the user wishes to select only certain items from his domains the text in the DIAL7 subfile is printed. This text explains the use of simple and compound conditions and defines the syntax to be used for specifying a simple condition.

5. Call the Specify Condition routine (Section 15.1 of this appendix) to carry out the dialogue which prompts the user to enter his conditions for data retrieval. From the user's responses the appropriate COBOL conditional clauses are generated.

6. Generate the COBOL statements which specify the actions to be taken when the condition is satisfied and when it is not.

7. Generate the final statements for the SELCOND macro, save them all in the SELCOND subfile and chain to the STAGE8 macro.

The output of the instructional text in step 4 could be made optional for more experienced users of the system (Section 8.4 of the main text).
15.1 SPECIFY CONDITION ROUTINE

This routine carries out the condition specifying dialogue which consists of the following tasks:

1. Generate the COBOL 'IF' word.

2. Ask the user if he wishes to specify a compound condition. If his reply is 'yes' continue processing at step 7 below.

3. Ask the user to enter the simple condition.

4. Issue the prompt and read the user's condition entry.

5. Call the Validate Condition routine (Section 15.2 of this appendix) to validate the simple condition. It will either generate the equivalent COBOL condition clause or print an error message and set a local variable error indicator equal to 1. In the latter case processing resumes at step 4 above, in the former case it continues below.

6. Exit from the dialogue routine.

7. Set a local text variable equal to 'FIRST'. This text variable is used in the condition request message and takes the value 'FIRST' or 'NEXT'.

8. Ask the user to enter the first or next condition as appropriate.

9. Issue the prompt and read the user's condition statement.

10. Call the Validate Condition routine to validate the condition. If an error is detected processing resumes at step 9 above, otherwise it continues below (c.f. step 5).

11. If the local text variable is equal to 'FIRST' continue at step 13, otherwise continue below.

12. Ask the user if he has any more clauses to enter. If the reply is 'no' resume processing at step 6 above, otherwise continue at step 14 below.

13. Change the contents of the local text variable to be
14. Ask the user to specify which conjunction, 'AND' or 'OR', precedes the next simple condition and generate a COBOL statement containing the selected conjunction. If the 'OR' conjunction is specified the user is reminded that he must re-enter any of the previously entered conditions which still apply (Section 5.5.3 of main text). Processing resumes at step 8 above.

15.2 VALIDATE CONDITION ROUTINE

This routine validates the user's simple condition statement which is located in the input buffer and from it generates the equivalent COBOL statement. The COBOL statement occupies two lines in order to allow for the qualification of data-names and takes the form:

```
operand-1 IN Z-TUPLE conditional-operator
operand-2 IN Z-TUPLE
```

The qualification in the second line is omitted if operand-2 is a literal.

When an error is detected a message is output on the user's terminal, the local variable used as an error indicator is set equal to 1 and control returns to the calling routine without the generation of the COBOL statement. An error recovery procedure is carried out in the Specify Condition routine, the calling routine, when the error indicator is found to be equal to 1.

The processing steps are as follows:

1. Set the local variable used as the error indicator equal to zero.

2. Validate the syntax of the simple conditional statement and separate the three components. The operands are stored in string variables and the operator is stored in a local variable. Processing and storage details are summarised in Table 15.1.
3. Validate the first operand which must be a domain or temporary item name. Processing details are given in Table 15.2.

4. Validate the operator and build up the equivalent COBOL operator in a string variable. Processing details are given in Table 15.3.

5. Save, in a local variable, the type of the first operand, either numeric or alphanumeric. The type is determined in step 3 above and is used again in step 8.

6. Determine the length of the second operand and place a copy of the operand into the input buffer ready for validation.

7. Identify the nature of the second operand by inspection of the first character of the input buffer and validate accordingly. Processing details are given in Tables 15.4 to 15.7.

8. Compare the type, alphanumeric or numeric, of the first and second operands. If they are not the same an error message is output on the user's terminal, the error indicator is set equal to 1 and processing continues at step 10.

9. Generate the COBOL condition statement using the preset contents of selected string variables.

10. Exit from the routine.

Table 15.1 Syntax validation for a Simple Condition located in the input buffer

<table>
<thead>
<tr>
<th>Conditions to be satisfied</th>
<th>Actions to be take when all conditions are satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Both full stop delimiters for the operator must be present.</td>
<td>1. Extract operand-1 and store in string variable #303.</td>
</tr>
<tr>
<td>2. The second full stop delimiter must not be the last non-blank character in the input buffer.</td>
<td>2. Extract the operator, omitting the delimiters, and store in local variable #LOP.</td>
</tr>
<tr>
<td></td>
<td>3. Extract operand-2 and store in string variable #304.</td>
</tr>
</tbody>
</table>
Table 15.2 Validate 1st Operand as a domain or temporary item name

<table>
<thead>
<tr>
<th>Actions to be taken before validation</th>
<th>Conditions to be satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Make a copy of the operand name in string variable $S09.</td>
<td>1. The binary search success indicator, $LIND, must be zero.</td>
</tr>
<tr>
<td>2. Binary search the DOMAININDEX subfile for a name matching that in $S09.</td>
<td></td>
</tr>
</tbody>
</table>

N.B. An extended version of the Search routine (Section 2 of this appendix) is used which enables the type of field to be extracted from the DOMAININDEX subfile record and stored in local variable $LTYPE.

Table 15.3 Validate Condition Operator

<table>
<thead>
<tr>
<th>Action to be taken before validation</th>
<th>Permitted values of condition operator in $LOP</th>
<th>Equivalent COBOL condition set up in string variable $S08</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Remove any imbedded blanks in the operator by squashing the contents of local variable $LOP.</td>
<td>EQ, LT, LE, GT, GE, NE, NOTEQ, NOTLT, NOTLE, NOTGT, NOTGE, NOTNE</td>
<td>EQUAL TO, LESS THAN, NOT GREATER THAN, GREATER THAN, NOT LESS THAN, NOT EQUAL TO, NOT EQUAL TO, NOT LESS THAN, GREATER THAN, NOT GREATER THAN, LESS THAN, EQUAL TO</td>
</tr>
</tbody>
</table>
### Table 15.4 Identify nature of Second Operand

<table>
<thead>
<tr>
<th>1st character</th>
<th>Operand type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>Alphanumeric literal</td>
</tr>
<tr>
<td>+ - . or digit</td>
<td>Numeric literal</td>
</tr>
<tr>
<td>Any other character</td>
<td>Domain or temporary item name</td>
</tr>
</tbody>
</table>

### Table 15.5 Validation of Alphanumeric literal 2nd Operand

<table>
<thead>
<tr>
<th>Conditions to be satisfied</th>
<th>Actions to be taken when all conditions are satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Length &gt; 3</td>
<td>Set local variable $#TYPE$ equal to 'A' to denote that the second operand is of alphanumeric type.</td>
</tr>
<tr>
<td>2. Last character a double quotation mark (&quot;').</td>
<td></td>
</tr>
</tbody>
</table>

### Table 15.6 Validation of Numeric literal 2nd Operand

<table>
<thead>
<tr>
<th>Conditions to be satisfied</th>
<th>Action to be taken when all conditions are satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Subsequent characters must be digits or a decimal point.</td>
<td>Set local variable $#TYPE$ equal to 'N' to denote that the second operand is of numeric type.</td>
</tr>
<tr>
<td>2. After a decimal point only digits are permitted.</td>
<td></td>
</tr>
<tr>
<td>3. The decimal point may not be the last character.</td>
<td></td>
</tr>
</tbody>
</table>
Table 15.7 Validation of Domain or Temporary item as 2nd Operand

<table>
<thead>
<tr>
<th>Actions to be taken before validation</th>
<th>Conditions to be satisfied</th>
<th>Action to be taken when conditions are satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Make a copy of the operand name in string variable #SO9</td>
<td>1. The binary search success indicator, #LIND, must be zero.</td>
<td>1. Append the qualification IN Z-TUPLE to the name of the second operand which is stored in string variable #SO4.</td>
</tr>
<tr>
<td>2. Binary search the DOMAININDEX subfile for a name matching that in #SO9, N.B. An extended version of the Search routine (Section 2 of this appendix) is used which enables the field type to be extracted from the DOMAININDEX subfile record and stored in local variable #LTYPE.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16. STAGE8 MACRO

The processing steps for the STAGE8 macro are shown below. On the macro listing (Item 27 in the separate folder) the first statement of each step has been highlighted.

1. Set the check point variable #GCKPT equal to 8.

2. Print the introductory dialogue for the stage and ask the user to select the report page width. Although a narrow page of 70 characters or a wide page of 120 characters is offered, only the implementation of the narrow page has been attempted. Global variable #GWDTH is used to store the user's option as a single character, 'N' for a narrow page or 'W' for a wide page. (A user selecting a wide page is told that the facility is not yet available and by default a narrow page is assumed.)

3. Ask the user to enter the maximum number of lines to be
printed on a report page. This must be a number in the range 40 to
99, with a default value of 60. Global variable &GXLN is used to
store in binary form the maximum number of lines on a report page.

4. Print the dialogue asking the user if he wishes to
design his own report line formats or to accept the default
formats generated by the system. (The default option has not yet
been implemented so anyone selecting this option is advised that
he must design his own output formats.)

5. Print the text describing how to select and associate
a label character with each domain or temporary item appearing
in the report.

6. Carry out the housekeeping operations necessary to
maintain strings of valid and unallocated label characters and
pointers to current records in the subfiles used by the macro.

7. Display, one by one, the names of the domains and
temporary items in the DOMAININDEX subfile and ask the user to
assign a unique label character to those which are to appear in
the printed report.

8. Use the data from the DOMAININDEX record to set up and
write the LABELTABLE record for the currently selected label
character.

9. Steps 7 and 8 are repeated until either all the names
in the DOMAININDEX have been displayed or all the label characters
have been allocated. In the latter case a warning message is
output.

10. Write LABELTABLE records for all unused label
characters, if any, so that all fields except the first are empty.

11. Print the remaining instructional text from the DIAL8
subfile. This explains how to specify a report line format using
label characters.

12. Chain to the STAGE9 macro.
17. STAGE9 MACRO NOTES

These notes should be read in conjunction with Figure 7.20 in the main text of the thesis. The note numbers refer to the block with the corresponding number in the figure.

1. The string of all valid label characters is set up in string variable $S03$.

2. The user is prompted to enter an alphanumeric editing format which is saved in string variable $S04$. The contents of $S04$ is validated according to the rules shown in Table 17.1. The user is then prompted to enter the alphanumeric value to be edited according to the previously defined format. The characters entered are transferred to the output buffer according to the rule shown in Table 17.2.

3. Prompted by the dialogue the user enters a numeric editing format which is stored in string variable $S04$ and validated according to the rules shown in Table 17.3. The numeric literal which the user is then prompted to enter must satisfy the conditions set out in Table 12.1 of this appendix. The edited form of the numeric literal is built up in the output buffer using the character manipulations described in Tables 17.4 to 17.6.

4. A copy of the check point variable $GCKPT$ is saved in local variable $LV1$. $GCKPT$ is then reduced by 1 so that, when control chains to the stage designated by $LV1$, the stage appears to have been entered normally.
Table 17.1 Validate Alphanumeric editing format

<table>
<thead>
<tr>
<th>Conditions to be satisfied by Alphanumeric editing format stored in string variable #S04</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The string of edit format characters must not contain imbedded blank or # characters, although it may begin with one or more # characters.</td>
</tr>
<tr>
<td>2. The label characters in the string must all be the same.</td>
</tr>
<tr>
<td>3. The label character must be one of the those contained in string variable #S03.</td>
</tr>
<tr>
<td>4. In addition to the label characters and any leading # characters, only B and O characters may be present in the editing format. If present, the B or O characters must appear somewhere to the right of a label character, but not in the last character position.</td>
</tr>
</tbody>
</table>

Table 17.2 Character transfers to the output buffer during an Alphanumeric Edit

<table>
<thead>
<tr>
<th>Edit characters from string variable #S04</th>
<th>Character deposited in the output buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>% (i.e. blank)</td>
</tr>
<tr>
<td>Label character</td>
<td>Next character from the alphanumeric value located in the input buffer. (The buffer is scanned from left to right.)</td>
</tr>
<tr>
<td>B</td>
<td>% (i.e. blank)</td>
</tr>
<tr>
<td>O</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 17.3 Validate Numeric editing format

<table>
<thead>
<tr>
<th>Conditions to be satisfied by Numeric editing format stored in string variable #S04</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The string of edit format characters must not contain imbedded blank or # characters, although it may begin with one or more # characters.</td>
</tr>
<tr>
<td>2. + or - may appear only once and may only be preceded by a # character.</td>
</tr>
<tr>
<td>3. % or £ may appear only once and may only be preceded by a # + or - character.</td>
</tr>
<tr>
<td>4. • may only appear once and may only be preceded by a # + - % or £ character.</td>
</tr>
<tr>
<td>5. The label characters in the string must all be the same.</td>
</tr>
<tr>
<td>6. The label character must be one of those contained in string variable #S03.</td>
</tr>
<tr>
<td>7. • may appear only once, but not as the last character in the string. It must be preceded by a least one label character.</td>
</tr>
<tr>
<td>8. CR and DB may appear as the last two characters of the edit format, but not if + or - have been used.</td>
</tr>
<tr>
<td>9. B , 0 characters may be present, but only if they are somewhere to the right of a label character and not in the last position of the edit string.</td>
</tr>
<tr>
<td>10. Excluding leading # characters the edit format string must not exceed 30 characters in length.</td>
</tr>
</tbody>
</table>
Table 17.4 Character transfers to the output buffer during a Numeric Edit - Part 1

<table>
<thead>
<tr>
<th>Characters to the left of the implied or specified decimal point, reading from right to left</th>
<th>Edit character from string variable #504</th>
<th>Character deposited in the output buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>Next digit from the numeric value located in the input buffer. (The buffer is scanned from right to left, starting with the character to the left of the implied or specified decimal point.) If the input buffer digits have all been processed or a sign character is reached, then a 0 character is used.</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>if the numeric value in the input buffer is positive or unsigned.</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>if the numeric value in the input buffer is negative.</td>
</tr>
<tr>
<td>£ or %</td>
<td>£ or %</td>
<td>if the numeric value in the input buffer is negative. % (i.e. blank) if the numeric value in the input buffer is positive or unsigned.</td>
</tr>
<tr>
<td>B</td>
<td>% (i.e. blank)</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>,</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>% (i.e. blank)</td>
<td></td>
</tr>
</tbody>
</table>
Table 17.5 Character transfers to the output buffer during a Numeric Edit - Part 2

<table>
<thead>
<tr>
<th>Edit character from string variable</th>
<th>Character(s) deposited in the output buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label character</td>
<td>Next character from the numeric value located in the input buffer. (The buffer is scanned from left to right, starting with the character to the right of the implied or specified decimal point.) If the character is blank, i.e. the input value characters are exhausted, a 0 character is used.</td>
</tr>
<tr>
<td>B</td>
<td>% (i.e. blank)</td>
</tr>
<tr>
<td>v</td>
<td>,</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CR or DB</td>
<td>If the input buffer contains a positive or unsigned value %% is used (i.e. 2 blanks). If the input buffer contains a negative value the CR or DB characters are used.</td>
</tr>
</tbody>
</table>
Table 17.6 Adjustments to the contents of the output buffer prior to printing edited numeric data

<table>
<thead>
<tr>
<th>Conditions to be satisfied by the contents of the output buffer</th>
<th>Action to be taken to amend the contents of the output buffer when the conditions are satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>None of the following edit characters present + - £ *</td>
<td>Replace all leading zeros, if any, up to but not including a zero immediately to the left of the implied or actual decimal point by blank characters.</td>
</tr>
<tr>
<td>* present</td>
<td>Replace all leading zeros, if any, to the right of the * character up to, but not including, a zero to the left of the implied or actual decimal point by a * character.</td>
</tr>
</tbody>
</table>
| £ or £ present but * absent                                  | 1. Replace all leading zeros, if any, to the right of the £ or € character up to, but not including, a zero to the left of the implied or actual decimal point by a blank.  
   2. Replace the £ or € character by a blank.  
   3. Replace the blank character to the left of the left-most digit by a £ or €. |
| + or - present but £ £ and * all absent                      | 1. Replace all leading zeros, if any, to the right of the + or - character up to, but not including, a zero to the left of the implied or actual decimal point by blank characters.  
   2. Replace the + or - character by a blank.  
   3. Replace the blank character to the left of the left-most digit by a + or - character. |
18. IDENTIFICATION, VALIDATION AND GENERATION OF FIELD DESCRIPTIONS

18.1 BLANK FILLER

Table 18.1 Identification and validation of Blank FILLER field

<table>
<thead>
<tr>
<th>Start of Field Identification</th>
<th>End of Field Identification</th>
<th>Field Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>First character #</td>
<td>Last character followed by a non # character</td>
<td>Only # characters present.</td>
</tr>
</tbody>
</table>

The COBOL data description statement generated takes the following form:

```
02 FILLER PICTURE X(n) VALUE SPACES.
```

where \( n \) is the number of contiguous \# characters in the field.

18.2 ALPHANUMERIC FILLER

Table 18.2 Identification and validation of Alphanumeric FILLER field

<table>
<thead>
<tr>
<th>Start of Field Identification</th>
<th>End of Field Identification</th>
<th>Field Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>First character &quot;</td>
<td>Last character &quot;</td>
<td>Both delimiting quotes (&quot;) must be present.</td>
</tr>
</tbody>
</table>

The COBOL data description statements generated take the following form:

```
02 FILLER PICTURE X(n) VALUE "text-characters".
```

where

\( n \) is equal to two plus the number of characters between the
delimiting quotation marks in the line specification, and

\[ \text{\textbullet} \]
is a space character to compensate for that occupied by a " character in the line specification.

If the number of characters in a text string is greater than 56, then statements for two FILLER items are generated as follows:

```plaintext
02 FILLER PICTURE X(57) VALUE "first-fiftysix-text-characters".
02 FILLER PICTURE X(m) VALUE "remaining-text-characters\textbullet".
```

where \( m = n - 57 \) and \( n \) is as described above. Again \( \text{\textbullet} \) denotes a compensating space character.

18.3 ALPHANUMERIC DATA ITEM

Table 18.3 Identification and validation of Alphanumeric data field

<table>
<thead>
<tr>
<th>Start of Field Identification</th>
<th>End of Field Identification</th>
<th>Field Validation</th>
</tr>
</thead>
</table>
| First character - valid label character whose LABELTABLE record contains details of an alphanumeric item, i.e. Field type equals 'A' | Last character followed by one of the following: 
- #
- +
- $
- @
- "
Alphabetic characters other than the current label character | 1. The label character must not previously have appeared in a field description in the current line.
2. B and 0 characters may be present, but only if they are somewhere to the right of a label character and not in the last character position of the field specification.
3. The resulting PICTURE string (Table 18.4) must not be more than 30 characters long.
The COBOL data description generated has the following form:

```
02 data-name PICTURE picture-string.
```

where

data-name is the second field of the LABELTABLE record for the

label character used in the field format description,

and

picture-string is the value of a string variable whose contents

is derived from the field format specification in the way

indicated in Table 18.4.

Table 18.4 Derivation of a picture string from an alphanumeric

field format specification.

<table>
<thead>
<tr>
<th>Alphanumeric field specification character(s)</th>
<th>Corresponding picture-string character(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label character</td>
<td>X</td>
</tr>
<tr>
<td>Five or more contiguous label characters</td>
<td>X(n) where n is the number of contiguous label characters.</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Five or more contiguous B characters</td>
<td>B(n) where n is the number of contiguous B characters.</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

N.B. The resulting picture-string must not exceed 30 characters.
### Table 18.5 Identification and validation of Numeric data field

<table>
<thead>
<tr>
<th>Start of Field Identification</th>
<th>End of Field Identification</th>
<th>Field Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First character</strong></td>
<td><strong>Last two characters</strong></td>
<td>1. At least one valid label character, whose LABELTABLE record contains details of a numeric item must be present.</td>
</tr>
<tr>
<td>one of the following:</td>
<td>CR or DB</td>
<td>2. The label character must not previously have appeared in a field description in the same line.</td>
</tr>
<tr>
<td>+</td>
<td>or</td>
<td>3. + or - may only appear as the first character of the field.</td>
</tr>
<tr>
<td>£</td>
<td>last character</td>
<td>4. £ or £ may only appear once and may only be preceded by a + or -.</td>
</tr>
<tr>
<td>*</td>
<td>followed by one of the following:</td>
<td>5. * may only appear once and may only be preceded by + - £ or £.</td>
</tr>
<tr>
<td>Valid label character whose LABELTABLE record contains details of a numeric item, i.e. Field type equals 'N'</td>
<td>£ or ; 2 &gt;  may only appear once and may only be preceded by + - £ or £.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* may only appear once and may only be preceded by + - £ or £.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>. may appear only once but not as the last character. It must be preceded by a label character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If used, CR or DB may only appear as the last two characters of the field. They may not be used when a + or - is present in the field specification.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B , 0 characters may be present, but only if they are somewhere to the right of a label character and not in the last position of the field specification.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30 characters is the maximum field specification size.</td>
<td></td>
</tr>
</tbody>
</table>
The COBOL data description statement generated takes the following form:

```
02 data-name PICTURE picture-string.
```

where

data-name is the second field of the LABELTABLE record for the label character in the field specification.

and

picture-string is the value of a string variable whose contents is derived from the field format specification in the way indicated in Tables 18.6 and 18.7.

Table 18.6 Derivation of a picture string from a numeric field format specification - Part 1

<table>
<thead>
<tr>
<th>Numeric field specification character(s)</th>
<th>Corresponding picture-string character(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label character</td>
<td>9</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>£ or €</td>
<td>£ or €</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>,</td>
<td>,</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>CR</td>
<td>CR</td>
</tr>
<tr>
<td>DB</td>
<td>DB</td>
</tr>
</tbody>
</table>
### Table 18.7 Derivation of a picture string from a numeric field format specification - Part 2

<table>
<thead>
<tr>
<th>Conditions to be satisfied by the picture-string</th>
<th>Changes to be made to the picture-string</th>
</tr>
</thead>
<tbody>
<tr>
<td>None of the following edit characters present:</td>
<td>Replace each 9 character to the left of that in the units position, if any, by a Z (i.e. the COBOL zero suppression character).</td>
</tr>
<tr>
<td>$ + - £ * $</td>
<td>Replace each 9 character to the right of the * and to the left of that in the units position, if any, by an * (i.e. the COBOL floating cheque protect character).</td>
</tr>
<tr>
<td>£ or $ present but * absent</td>
<td>Replace each 9 character to the right of the £ or $ and to the left of the units position, if any, by a £ or $ (i.e. the COBOL floating currency character).</td>
</tr>
<tr>
<td>+ or - present but £ or $ or * absent</td>
<td>Replace each 9 character to the right of the + or - and to the left of the units position, if any, by a + or - (i.e. the COBOL floating sign character).</td>
</tr>
</tbody>
</table>

#### 18.5 DATE ITEM

The date field is identified in the line format specification by the presence of the eight character string DD/MM/YY. It may only occur once in any line. The COBOL level 02 data description generated has the following form:

```
02 ZDATE PICTURE X(8).
```
18.6 TIME ITEM

The eight character string HH/MM/SS identifies the presence of the time field in the line format specification. It may occur only once in any line. The COBOL level 02 data description generated takes the form:

02 ZTIME PICTURE X(8).

18.7 PAGE NUMBER ITEM

The presence of the page number field in a line format specification is identified by the four character string PPPP. It may occur only once in any line. The COBOL level 02 statement generated has the following form:

02 Z-PAGE-COUNT PICTURE ZZZ9.
19. WRITEPARA MACRO

The general form of the statements generated during Stage 10 of the problem specifying dialogue for the WRITEPARA macro is as follows:

\%
\%DEF WRITEPARA
\% LABELOFF
\% IF Z-LINE-COUNT GREATER THAN Z-L-C(1) PERFORM Z-PARA-PAGE.
\% LABELON
\%END

where:

l equals the number of key domains, i.e. the value in global variable $\#NKEY$.

m equals one plus the number of blank lines before the first non-blank detail line of the print group.

n equals the non-blank detail line number. n takes the values 2, 3, 4, etc depending on the number of non-blank lines in the print group.

p equals one plus the number of blank lines before the current non-blank line of the print group.

r equals the total number of lines, including blank lines, in the detail print group.
20. TITLEPARA MACRO

The general form of the TITLEPARA macro, the statements for which are generated during Stage 12 of the problem specifying dialogue, is as follows:

```%
%DEF TITLEPARA
% LABELOFF
Z-PARA-TITLE.
MOVE SPACES TO Z-OUTREC.
WRITE Z-OUTREC BEFORE ADVANCING CHANNEL-1.
```

```
\begin{enumerate}
\item For first non-blank title line:
  \begin{enumerate}
  \item MOVE CORRESPONDING Z-TUPLE TO Z-TITLE-1.
  \item WRITE Z-OUTREC FROM Z-TITLE-1 AFTER ADVANCING 1 LINES.
  \end{enumerate}
\item For subsequent non-blank title lines, if any:
  \begin{enumerate}
  \item MOVE CORRESPONDING Z-TUPLE TO Z-TITLE-n.
  \item WRITE Z-OUTREC FROM Z-TITLE-n AFTER ADVANCING m LINES.
  \end{enumerate}
\end{enumerate}
```

```%
%LABELON
%END
```

where:

- \( n \) equals the number of the non-blank line. \( n \) takes the values 2, 3, etc depending on the number of non-blank lines in the print group.
- \( l \) equals the number of blank lines before the first non-blank title line.
- \( m \) equals one plus the number of blank lines before the current non-blank title line.

The MOVE CORRESPONDING statement before each WRITE operation is generated only if the date, time or page number appears in the line format specification.
21. PHEADPARA MACRO

The general form of the statements generated during Stage 13 of the problem specifying dialogue for the PHEADPARA macro is as follows:

\[ \% \text{DEF PHEADPARA} \]
\[ \% \text{LABELOFF} \]
\[ \text{MOVE CORRESPONDING Z-TUPLE TO Z-PAGE-1.} \]
\[ \text{WRITE Z-OUTREC FROM Z-PAGE-1 AFTER ADVANCING 1 LINES.} \]
\[ \text{MOVE CORRESPONDING Z-TUPLE TO Z-PAGE-n.} \]
\[ \text{WRITE Z-OUTREC FROM Z-PAGE-n AFTER ADVANCING m LINES.} \]
\[ \text{MOVE q TO Z-ADV-LINES.} \]
\[ \text{MOVE r TO Z-LINE-COUNT.} \]
\[ \text{MOVE 1 TO Z-PAGE-SWITCH} \]
\[ \% \text{LABELON} \]
\[ \% \text{END} \]

where:

- \( l \) equals the number of blank lines before the first non-blank page heading line.
- \( m \) equals one plus the number of blank lines before the current non-blank page heading line.
- \( n \) equals the number of the non-blank page heading line, i.e. \( n \) takes the values 2, 3, etc depending on the number of non-blank lines in the page heading.
- \( q \) equals the number of blank lines after the final non-blank page heading line.
- \( r \) equals the total number of lines, including blank lines, in the page heading print group.

The MOVE CORRESPONDING statement before each WRITE operation is generated only if the date, time, page number or a label character appears in the line format specification. If the user does not wish to specify a printed page heading, only the page control statements are generated in the body of the macro. In this case both \( q \) and \( r \) have the value zero.
for the SBHPARAn macros is as follows:

\[
\%\text{DEF} \quad \text{SBHPARAn} \\
\% \quad \text{LABELOFF} \\
\text{1 non-blank line of the sequence break heading} \\
\text{ADD 1 TO Z-ADV-LINES.} \\
\text{MOVE CORRESPONDING Z-TUPLE TO Z-HEAD}n-1. \\
\text{WRITE Z-OUTREC FROM Z-HEAD}n-1 \text{ AFTER ADVANCING Z-ADV-LINES.} \\
\text{Subsequent non-blank lines, if any} \\
\text{MOVE CORRESPONDING Z-TUPLE TO Z-HEAD}n-m. \\
\text{WRITE Z-OUTREC FROM Z-HEAD}n-m \text{ AFTER ADVANCING } p \text{ LINES.} \\
\% \quad \text{LABELON} \\
\%\text{END}
\]

where:

- \( l \) equals one plus the number of blank lines before the first non-blank line of the sequence break heading.
- \( m \) equals the number of the non-blank line in the sequence break heading, i.e. \( m \) takes the values 2, 3, etc depending on the number of non-blank lines in the print group.
- \( n \) equals the level number of the sequence key, i.e. \( n \) takes the values 1, 2, 3, etc depending on the number of keys. There is a sequence break heading for each key except the minor key and 1 denotes the major key.
- \( p \) equals one plus the number of blank lines before the current non-blank line of the sequence break heading.
- \( q \) equals the number of blank lines after the final non-blank line in the sequence break heading.
- \( r \) equals the total number of lines, including blank lines, in the current sequence break heading.

The MOVE CORRESPONDING statement is generated only if a label character appears in the format specification of the line.
The STLPARAn macros generated during Stage 15 have the following general form:

```%DEF STLPARAn
% LABELON
1st non-blank line of sequence break subtotal.
ADD 1 TO Z-ADV-LINES.
MOVE CORRESPONDING Z-LEVEL-n TO Z-SUBTOTALn-1.
MOVE data-name IN Z-TUPLE TO data-name IN
Z-SUBTOTALn-1.
WRITE Z-OUTREC FROM Z-SUBTOTALn-1 AFTER Z-ADV-LINES.
Subsequent non-blank lines, if any.
MOVE CORRESPONDING Z-LEVEL-n TO Z-SUBTOTALn-m.
MOVE data-name IN Z-TUPLE TO data-name IN
Z-SUBTOTALn-m.
WRITE Z-OUTREC FROM Z-SUBTOTALn-m AFTER ADVANCING p LINES.
MOVE q TO Z-ADV-LINES.
ADD r TO Z-LINE-COUNT.
% LABELON
%END
```

where:

- **l** equals one plus the number of blank lines before the first non-blank line of the sequence break subtotal.
- **m** equals the number of the non-blank line in the print group, i.e. m takes the values 2, 3, etc depending on the number of non-blank lines in the subtotal output.
- **n** equals the level number of the sequence key, i.e. n takes the values 1, 2, 3, etc depending on the number of keys. There is a sequence break subtotal for each key level except the minor key and 1 denotes the major key.
- **p** equals one plus the number of blank lines before the current non-blank line in the sequence break subtotal.
- **q** equals the number of blank lines after the final non-blank line of the print group.
- **r** equals the total number of lines, including blank lines, in the current sequence break subtotal.
data-name equals the name of a non-totalling item which appears in
the current line of the subtotal print group.

The MOVE CORRESPONDING statement is generated only if a label
caracter for a totalling item appears in the format specification
of the line. If only non-totalling data items are present in the
line format specification, the MOVE CORRESPONDING Z-LEVEL-n
statement is omitted and a MOVE CORRESPONDING Z-TUPLE statement
may be used to replace the individual MOVE statements.

24. TOTALPARA MACRO

The general form of the statements generated for the TOTALPARA
macro in Stage 16 is as follows:

```assembly
%DEF TOTALPARA
% LABELOFF
1 non-blank totals line.
ADD 1 TO Z-ADV-LINES.
MOVE CORRESPONDING Z-LEVEL-0 TO Z-TOTAL-1.
MOVE data-name IN Z-TUPLE TO data-name IN Z-TOTAL-1.
WRITE Z-OUTREC FROM Z-TOTAL-1 AFTER ADVANCING Z-ADV-LINES.

Subsequent non-blank totals lines if any.
MOVE CORRESPONDING Z-LEVEL-0 TO Z-TOTAL-m.
MOVE data-name IN Z-TUPLE TO data-name IN Z-TOTAL-m.
WRITE Z-OUTREC FROM Z-TOTAL-m AFTER ADVANCING p LINES.

% LABELON
% END
```

where:

l equals the number of blank lines before the first non-blank
line of the totals print group.

m equals the number of the non-blank lines in the totals print
group, i.e. m takes the values 2, 3, etc depending on the
number of non-blank lines in the totals output.

p equals one plus the number of blank lines before the current
non-blank line in the totals output.

data-name equals the name of a non-totalling item which appears in
in the current non-blank line of the totals print group.
The MOVE CORRESPONDING Z-LEVEL=0 statement is generated only if a
label character for a totalling item appears in the format
specification of the line. A MOVE data-name IN Z-TUPLE statement
is generated for each non-totalling item whose label character
appears in the format specification of the line. If only
non-totalling data items are present in a total line format
specification, the MOVE CORRESPONDING Z-LEVEL=0 statement is not
generated. A MOVE CORRESPONDING Z-TUPLE statement may then be used
to replace the individual MOVE statements, if any.
# COMBINATIONS OF REPORT OUTPUT CATEGORIES AND ASSOCIATED PAGE SIZE CONDITIONS

The combinations of report output categories and associated page size conditions are summarised in Table 25.1.

Table 25.1 Page size conditions

<table>
<thead>
<tr>
<th>Page Type</th>
<th>Categories of output which may be included</th>
<th>Global variable used to indicate output category present</th>
<th>Name of subfile containing line count details</th>
<th>Symbol used for line count value</th>
<th>Conditions to be satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title page</td>
<td>Report title</td>
<td>#G TITL</td>
<td>Pagetitle</td>
<td>Lr</td>
<td>Lr &lt; #GMLN</td>
</tr>
</tbody>
</table>
| Report page | Page heading | #GPAGE | Pagehead | Lp | L+Ld < #GMLN 
| | Sequence break heading | #GHEAD | Pagesbh where n=1 to m and m=#GKEY -1 | Ls where L=Lp+ Σ Lsh | |
| | Detail line(s) | Always present | Pagedetail | Ld | |
| | Page heading | #GPAGE | Pagehead | Lp | L+Lst; < #GMLN 
| | Sequence break heading | #GHEAD | Pagesbh where n=1 to m and m=#GKEY -1 | Ls where L=Lp+ Σ Lsh | |
| | Sequence break subtotal | #GSUBT | Pagesst where j indicates the key level | Latj | |

N.B. Global variable #GMLN contains the maximum number of lines permitted on a page.
### Table 25.1 continued

<table>
<thead>
<tr>
<th>Page Type</th>
<th>Categories of output which may be included</th>
<th>Global variable used to indicate output category present</th>
<th>Name of subfile containing line count details</th>
<th>Symbol used for line count value</th>
<th>Conditions to be satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report page</td>
<td>Page heading</td>
<td>$#GPAGE$</td>
<td>PAGEHEAD</td>
<td>$L_p$</td>
<td>$L + L_t \leq #GMXLN$ where $L = L_p + \sum L_{sh_i}$ and $m = #GKEY - 1$</td>
</tr>
<tr>
<td></td>
<td>Sequence break heading</td>
<td>$#GHEAD$</td>
<td>PAGESBHn where $n=1$ to $m$ and $m=#GKEY - 1$</td>
<td>$L_{sh_i}$</td>
<td>$L_t$</td>
</tr>
<tr>
<td></td>
<td>Totals</td>
<td>$#GTOTL$</td>
<td>PAGETOTL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**N.B.** Global variable $\#GMXLN$ contains the maximum number of lines permitted on a page.
26. OWNname MACROS

The general form of the OWNname macros is as follows:

```
%DEF OWNname
% LABELOFF
conditional-expression-1
imperative-statement-1.
conditional-expression-2
imperative-statement-2.
; ;
% LABELON
%END
```

where:

name is the processing point in the COBOL program where the own code has to be executed (See Table 26.1 of this appendix for details).

conditional-expression represents lines of COBOL generated from the user entered condition specification which may be simple or compound. Condition specification is optional.

imperative-statement represents the COBOL statement generated from the user's own code statement (Section 27 of this appendix).
Table 26.1 Macro definition subfile names used for own code processing

<table>
<thead>
<tr>
<th>Subfile name</th>
<th>COBOL program processing point</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNSTART</td>
<td>At the start of processing before any data is retrieved.</td>
</tr>
<tr>
<td>OWNRETR</td>
<td>Immediately after a data retrieval.</td>
</tr>
<tr>
<td>OWNITL</td>
<td>Prior to printing the report title.</td>
</tr>
<tr>
<td>OWSBK1</td>
<td>At a sequence break in the key domain whose level is indicated by the digit at the end of the subfile name. Level 1 is for the major key.</td>
</tr>
<tr>
<td>OWSBK2</td>
<td></td>
</tr>
<tr>
<td>OWSBK3</td>
<td></td>
</tr>
<tr>
<td>OWSBK4</td>
<td></td>
</tr>
<tr>
<td>OWSBK5</td>
<td>N.B. In any problem only up to m subfiles will be used where ( m = #\text{NKEY}-1 )</td>
</tr>
<tr>
<td>OWSBK6</td>
<td></td>
</tr>
<tr>
<td>OWSBK7</td>
<td></td>
</tr>
<tr>
<td>OWSBK8</td>
<td></td>
</tr>
<tr>
<td>OWSBK9</td>
<td></td>
</tr>
<tr>
<td>OWSBK10</td>
<td></td>
</tr>
<tr>
<td>OWSBK11</td>
<td></td>
</tr>
<tr>
<td>OWNPAK</td>
<td>Prior to printing a page heading.</td>
</tr>
<tr>
<td>OWSBH1</td>
<td>Prior to printing the sequence break heading for the key domain whose level is indicated by the digit at the end of the subfile name. Level 1 is for the major key.</td>
</tr>
<tr>
<td>OWSBH2</td>
<td></td>
</tr>
<tr>
<td>OWSBH3</td>
<td></td>
</tr>
<tr>
<td>OWSBH4</td>
<td></td>
</tr>
<tr>
<td>OWSBH5</td>
<td>N.B. In any problem only up to m subfiles will be used where ( m = #\text{NKEY}-1 )</td>
</tr>
<tr>
<td>OWSBH6</td>
<td></td>
</tr>
<tr>
<td>OWSBH7</td>
<td></td>
</tr>
<tr>
<td>OWSBH8</td>
<td></td>
</tr>
<tr>
<td>OWNDETAIL</td>
<td>Prior to printing the detail line(s).</td>
</tr>
<tr>
<td>OWNSTL1</td>
<td>Prior to printing the sequence break subtotals for the sequence key whose level is indicated by the digit at the end of the subfile name. The major key is at level 1.</td>
</tr>
<tr>
<td>OWNSTL2</td>
<td></td>
</tr>
<tr>
<td>OWNSTL3</td>
<td></td>
</tr>
<tr>
<td>OWNSTL4</td>
<td></td>
</tr>
<tr>
<td>OWNSTL5</td>
<td></td>
</tr>
<tr>
<td>OWNSTL6</td>
<td></td>
</tr>
<tr>
<td>OWNSTL7</td>
<td>N.B. In any problem only up to m subfiles will be used where ( m = #\text{NKEY}-1 )</td>
</tr>
<tr>
<td>OWNSTL8</td>
<td></td>
</tr>
<tr>
<td>OWNTOTL</td>
<td>Prior to printing the total line(s).</td>
</tr>
</tbody>
</table>
27. OWN CODE STATEMENTS

This section describes the identification and validation of the user's own code statements from which equivalent COBOL statements are generated. The own code statements are described in the following order:

1. ADD
2. SUBTRACT
3. MULTIPLY
4. DIVIDE
5. MOVE

27.1 ADD OWN CODE STATEMENT

ADD operand-1, operand-2 GIVING operand-3

Table 27.1 Identification of the ADD statement and delimiters of operands

<table>
<thead>
<tr>
<th>Item</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Must be the first non-blank substring of the user's response.</td>
</tr>
<tr>
<td>operand-1</td>
<td>Starts with the first non-blank character after the final D of ADD.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank or comma character.</td>
</tr>
<tr>
<td>,</td>
<td>Must be present after the first operand but may be preceded by one or more blanks.</td>
</tr>
<tr>
<td>operand-2</td>
<td>Starts with the first non-blank character after the commas.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or by the first G of GIVING.</td>
</tr>
<tr>
<td>GIVING</td>
<td>Must be present after the second operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-3</td>
<td>Starts with the first non-blank character after the final G of GIVING.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the end of the response string.</td>
</tr>
</tbody>
</table>
Table 27.2 Permitted operand types for ADD, SUBTRACT and MULTIPLY

<table>
<thead>
<tr>
<th>Operand</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Numeric literal</td>
</tr>
<tr>
<td></td>
<td>(Note 1)</td>
</tr>
<tr>
<td>operand-1</td>
<td>✓</td>
</tr>
<tr>
<td>operand-2</td>
<td>✓</td>
</tr>
<tr>
<td>operand-3</td>
<td>✗</td>
</tr>
</tbody>
</table>

Notes

1. The rules for a valid numeric literal are given in Table 12.1 of this appendix.

2. The domain or temporary item name must have a matching record in the DOMAININDEX subfile with Field type equal to 'N' (Appendix V Section 7).

3. The domain or temporary item name must have a matching record in the DOMAININDEX subfile with the Totalling marker equal to 'T' or 'S'.

Three lines of COBOL are generated for each ADD own code statement and they take the following form:

ADD operand-1-replacement, operand-2-replacement GIVING operand-3-replacement.

where the operand replacement details are given in Table 27.3.
## Table 27.3 Operand replacement details for Arithmetic operations

<table>
<thead>
<tr>
<th>Own code operand type</th>
<th>COBOL replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric literal</td>
<td>Numeric literal</td>
</tr>
<tr>
<td>Domain or Temporary item name</td>
<td>item-name IN Z-TUPLE where item-name is that specified in the user's own code statement.</td>
</tr>
<tr>
<td>Domain or Temporary item name prefaced by *</td>
<td>item-name IN Z-LEVEL-n where item-name is that specified in the user's own code statement and n is the level number of the current sequence break key.</td>
</tr>
</tbody>
</table>

### 27.2 SUBTRACT OWN CODE STATEMENT

**SUBTRACT operand-1 FROM operand-2 GIVING operand-3**

Table 27.4 gives details of the identification of the SUBTRACT statement and the delimiters of the operands. The operand types permitted are shown in Table 27.2.

Three lines of COBOL are generated for each SUBTRACT own code statement and they take the following form:

**SUBTRACT operand-1-replacement FROM operand-2-replacement GIVING operand-3-replacement.**

The operand replacement details are given in Table 27.3.
Table 27.4 Identification of the SUBTRACT statement and delimiters of operands

<table>
<thead>
<tr>
<th>Item</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBTRACT</td>
<td>Must be the first non-blank substring of the user's response.</td>
</tr>
<tr>
<td>operand-1</td>
<td>Starts with the first non-blank character after the final T of SUBTRACT.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the F of FROM.</td>
</tr>
<tr>
<td>FROM</td>
<td>Must be present after the first operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-2</td>
<td>Starts with the first non-blank character after the M of FROM.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the first G of GIVING.</td>
</tr>
<tr>
<td>GIVING</td>
<td>Must be present after the second operand but may be preceded by one or more blanks.</td>
</tr>
<tr>
<td>operand-3</td>
<td>Starts with the first non-blank character after the final G of GIVING.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the end of the response string.</td>
</tr>
</tbody>
</table>

27.3 MULTIPLY OWN CODE STATEMENT

MULTIPLY operand-1 BY operand-2 GIVING operand-3

Table 27.5 gives details of the identification of the MULTIPLY statement and the delimiters of the operands. The operand types permitted are shown in Table 27.2.

Three lines of COBOL are generated for each MULTIPLY own code statement and they take the following form:

MULTIPLY operand-1-replacement BY operand-2-replacement GIVING operand-3-replacement.

The operand replacement details are given in Table 27.3.
### Table 27.5 Identification of the MULTIPLY statement and delimiters of operands

<table>
<thead>
<tr>
<th>Item</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLY</td>
<td>Must be the first non-blank substring of the user's response.</td>
</tr>
<tr>
<td>operand-1</td>
<td>Starts with the first non-blank character after the Y of MULTIPLY. Delimited by a blank character or the B of BY.</td>
</tr>
<tr>
<td>BY</td>
<td>Must be present after the first operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-2</td>
<td>Starts with the first non-blank character after the Y of BY. Delimited by a blank character or the first G of GIVING.</td>
</tr>
<tr>
<td>GIVING</td>
<td>Must be present after the second operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-3</td>
<td>Starts with the first non-blank character after the final G of GIVING. Delimited by a blank character or the end of the response string.</td>
</tr>
</tbody>
</table>
27.4 DIVIDE OWN CODE STATEMENT

DIVIDE operand-1 BY operand-2 GIVING operand-3

The remainder, if retained by the user, is operand-4.

Table 27.6 Identification of the DIVIDE statement and delimiters of operands

<table>
<thead>
<tr>
<th>Item</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVIDE</td>
<td>Must be the first non-blank substring of the user's response.</td>
</tr>
<tr>
<td>operand-1</td>
<td>Starts with the first non-blank character after the E of DIVIDE.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the B of BY.</td>
</tr>
<tr>
<td>BY</td>
<td>Must be present after the first operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-2</td>
<td>Starts with the first non-blank character after the Y of BY.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the first G of GIVING.</td>
</tr>
<tr>
<td>GIVING</td>
<td>Must be present after the second operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-3</td>
<td>Starts with the first non-blank character after the final G of GIVING.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the end of the response string.</td>
</tr>
<tr>
<td>operand-4</td>
<td>This exists only if the user enters a non-blank response when invited to save the remainder.</td>
</tr>
</tbody>
</table>

Table 27.7 Permitted operand types for DIVIDE

<table>
<thead>
<tr>
<th>Operand</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Numeric literal</td>
</tr>
<tr>
<td></td>
<td>(Note 1)</td>
</tr>
<tr>
<td>operand-1</td>
<td>✓</td>
</tr>
<tr>
<td>operand-2</td>
<td>✓</td>
</tr>
<tr>
<td>operand-3</td>
<td>×</td>
</tr>
<tr>
<td>operand-4</td>
<td>×</td>
</tr>
<tr>
<td>(optional)</td>
<td>×</td>
</tr>
</tbody>
</table>
Notes

1. The rules for a valid numeric literal are given in Table 12.1 of this appendix.

2. The domain or temporary item name must have a matching record in the DOMAININDEX subfile with Field type equal to 'N' (Appendix V Section ?).

3. The domain or temporary item name must have a matching record in the DOMAININDEX subfile with the Totalling marker equal to 'T' or 'S'.

Four or three lines of COBOL are generated for each DIVIDE own code statement entered by the user. Four lines are generated if the user opts to retain the remainder, i.e. operand-4 is specified, otherwise the three line version is generated. These statements have the following forms:

```
DIVIDE operand-1-replacement BY
operand-2-replacement GIVING
operand-3-replacement
REMAINDER operand-4-replacement.
```

or

```
DIVIDE operand-1-replacement BY
operand-2-replacement GIVING
operand-3 replacement.
```
Table 27.8 Identification of the MOVE statement and delimiters of operands

<table>
<thead>
<tr>
<th>Item</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>Must be the first non-blank substring of the user's response.</td>
</tr>
<tr>
<td>operand-1</td>
<td>Starts with the first non-blank character after the E of MOVE.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the T of TO.</td>
</tr>
<tr>
<td>TO</td>
<td>Must be present after the first operand but may be preceded by one or more blank characters.</td>
</tr>
<tr>
<td>operand-2</td>
<td>Starts with the first non-blank character after the O of TO.</td>
</tr>
<tr>
<td></td>
<td>Delimited by a blank character or the end of the response string.</td>
</tr>
</tbody>
</table>

Table 27.9 Permitted operand types for MOVE

<table>
<thead>
<tr>
<th>Operand</th>
<th>Type</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Numeric literal (Note 1)</td>
<td>Alphanumeric</td>
<td>Domain or Temporary item name (Note 3)</td>
</tr>
<tr>
<td>operand-1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(Note 4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operand-2</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

Notes

1. The rules for a valid numeric literal are given in Table 27.1 of this appendix.

2. An alphanumeric literal is delimited by double quotation mark characters (").

3. The domain or temporary item name must have a matching
4. Operand-1 must be of the same type, i.e. numeric or alphanumeric, as operand-2. When an operand is a domain or temporary item its type is indicated in its DOMAININDEX subfile record.

Two lines of COBOL are generated for each MOVE own code statement entered by the user and they take the following form:

```
MOVE operand-1-replacement
TO operand-2-replacement.
```

where the operand replacement details are given in Table 27.10.

Table 27.10 Operand replacement details for MOVE operations

<table>
<thead>
<tr>
<th>Own code operand type</th>
<th>COBOL replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric literal</td>
<td>Numeric literal</td>
</tr>
<tr>
<td>Alphanumeric literal</td>
<td>Alphanumeric literal</td>
</tr>
<tr>
<td>Domain or Temporary item name</td>
<td>item-name IN Z-TUPLE</td>
</tr>
<tr>
<td></td>
<td>where item-name is that specified in the user's own code statement.</td>
</tr>
</tbody>
</table>
The statements generated for the CONSTANTS macro have the following form:

\[
\begin{align*}
\% &\text{DEF CONSTANTS} \\
&\% \text{RESET } \#05 \\
&\% \text{DEPOSIT} \quad \ldots \quad \ldots \\
\% &\#GNREL = \ldots \\
\% &\#GNKEY = \ldots \\
\% &\#GTOT = \ldots \\
\end{align*}
\]

\%END

Entries for all the global variables shown in Table 28.1

### Table 28.1 String and global variables in the CONSTANTS macro

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Stage in which first used</th>
<th>Type of data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>#05</td>
<td>6</td>
<td>C</td>
<td>Message for file mismatch condition. Only present if #GMOP is equal to 'Y'</td>
</tr>
<tr>
<td>#GNREL</td>
<td>2</td>
<td>B</td>
<td>No. of relations (files) used.</td>
</tr>
<tr>
<td>#GNKEY</td>
<td>3</td>
<td>B</td>
<td>No. of sequence keys.</td>
</tr>
<tr>
<td>#GTOT</td>
<td>4</td>
<td>B</td>
<td>Totalling indicator 1 = totals required 0 = no totals required</td>
</tr>
<tr>
<td>#GDMCT</td>
<td>5</td>
<td>B</td>
<td>No. of domains in DOMAININDEX subfile.</td>
</tr>
<tr>
<td>#GMOP</td>
<td>6</td>
<td>C</td>
<td>Message option indicator Y = message required N or blank = no message</td>
</tr>
<tr>
<td>#GWIDTH</td>
<td>8</td>
<td>C</td>
<td>Page width indicator N = narrow page W = wide page</td>
</tr>
<tr>
<td>#GMXLN</td>
<td>8</td>
<td>B</td>
<td>Maximum no. of lines on a page.</td>
</tr>
<tr>
<td>#GTITL</td>
<td>12</td>
<td>B</td>
<td>1 = Title page present 0 = Title page absent</td>
</tr>
</tbody>
</table>
Table 28.1 continued

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Stage in which first used</th>
<th>Type of data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>#GPAGE</td>
<td>13</td>
<td>B</td>
<td>1 = Page heading present 0 = Page heading absent</td>
</tr>
<tr>
<td>#GHEAD</td>
<td>14</td>
<td>B</td>
<td>1 = Sequence break headings present 0 = Sequence break headings absent</td>
</tr>
<tr>
<td>#GSUBT</td>
<td>15</td>
<td>B</td>
<td>1 = Sequence break subtotals present 0 = Sequence break subtotals absent</td>
</tr>
<tr>
<td>#GTOTL</td>
<td>16</td>
<td>B</td>
<td>1 = Totals output required 0 = Totals output not required</td>
</tr>
</tbody>
</table>

29. HELP MACRO

The processing within the HELP macro consists of the steps outlined below. The first statement of each step has been highlighted on the macro listing (Item 28 in the separate folder).

1. Clean up the output stack by emptying any statements it may contain onto the WORK subfile which acts as a 'sink' (c.f Section 7.17 of main text).

2. Test the value of the check point global variable #GCKPT to see how many options should be offered to the user. Options 1 to 3 are always offered, but option 4 (trial editing) is offered only after Stage 9 of the problem specifying dialogue has been reached.

3. Output on the user's terminal the text from the DIALHELP subfile which lists the options available.

4. Invite the user to select an option.
5. Read and validate the user's response. This must be an integer in the range 1 to 3 if $GCKPT$ was less than 9, otherwise it may be in the range 1 to 4. The default response option is 1. If the response is invalid an error message is output on the user's terminal and the step is repeated.

6. Depending on the option value, processing continues at step 7, 8, 13 or 15.

7. Option 1. The 'Run abandoned' message is output on the user's terminal together with an instruction to type a $\%FINISH$ response. The $\%FINISH$ response effects an exit from the PG/2 macro processor system.

8. Option 2. A list of the stages at which the user may restart processing is output on his terminal from the DIALHELP text subfile. The value of the $GCKPT$ check point variable is used to determine the text lines printed and consequently the stages included in the option list.

9. Invite the user to enter the number of the stage at which he wishes to resume processing.

10. Read and validate the user's response. The response must consist of 1 or 2 digit characters which, when converted to binary, have a value in the range 1 to $GCKPT$. If the response is invalid an error message is output on the user's terminal and the step is repeated. The restart stage number is stored in local variable $\$LV1$.

11. The check point global variable $GCKPT$ is set equal to 1 less than the restart stage number. This ensures that the restart stage macro is entered normally, i.e. it appears to have been entered from the previous stage.

12. Chain to the macro for the selected restart stage. When this macro is not on the same mainfile as the HELP macro it is necessary to execute a MAINFILE macro-time statement to set
the appropriate mainfile prior to chaining to the selected macro.

13. Option 3. The text from the MODEL subfile with the
suffix value equal to $GCKPT$ is output on the user's terminal.
If $GCKPT$ is equal to 17 (Sample page stage) then a special
routine is used for outputting the text. This routine contains
macro-time statements which replace the greater than (>)
character at the beginning of any text record by a blank character prior
to its output on the user's terminal. These greater than characters
were inserted in otherwise blank records to overcome the difficulty
of reading all blank records with the FREAD macro-time statement
(Section 9.2 of Appendix V).

14. Set the restart stage number stored in local variable
$LV1$ equal to $GCKPT$ and continue processing at step 11 above.

15. Option 4. Chain to the STAGE9 macro which carries out
the dialogue for experimenting with the edit facilities.

30. SUMMARY OF SUBFILE USAGE AND INITIALISATION

Table 30.1 summarises the names and types of subfile used
by the COBOL report program generating system and shows their
arrangement on PG/2 macro processor mainfiles.

A WORK subfile is included in each mainfile where it is used
as the output subfile when macro and text subfiles are edited
during development (Appendix I Section1). The WORK subfile
containing the amended macro definition or text is then copied
back into the original subfile. The WORK subfile is also available
for use as a 'sink' by the HELP and report format specification
macros.

Some of the common data and 'grown' macro definition subfiles
require initialisation prior to the execution of the first stage
of the problem specifying dialogue. The initialisation is
achieved by using a preset input file containing the necessary
PG/2 macro processor commands which also contain the commands which load and call the STAGE1 macro. Further input is obtained from the user's terminal in response to the dialogue prompts. Details of the subfiles which require initialisation are given in Tables 30.2 and 30.3.
### Table 30.1 Subfile organisation

<table>
<thead>
<tr>
<th>Mainfile name</th>
<th>GWAC132-DATA</th>
<th>GWAC132-DAT1</th>
<th>GWAC132-DAT2</th>
<th>GWAC132-DAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Subfile Type</td>
<td>Subfile Type</td>
<td>Subfile Type</td>
<td>Subfile Type</td>
</tr>
<tr>
<td>1</td>
<td>CREATE</td>
<td>M</td>
<td>CONCAT</td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>MESSAGES</td>
<td>T</td>
<td>TEMPTITEM</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>PASSCHANGE</td>
<td>M</td>
<td>STAGE6</td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>LIBRARY</td>
<td>M</td>
<td>DIAL6</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>LIBPRELIM</td>
<td>M</td>
<td>STAGE7</td>
<td>M</td>
</tr>
<tr>
<td>6</td>
<td>LIBADD</td>
<td>M</td>
<td>DIAL7</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>LIBDELETE</td>
<td>M</td>
<td>SELCOND</td>
<td>M</td>
</tr>
<tr>
<td>8</td>
<td>CATALOGUE</td>
<td>C</td>
<td>STAGE8</td>
<td>M</td>
</tr>
<tr>
<td>9</td>
<td>STAGE1</td>
<td>M</td>
<td>DIAL8</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>DIAL1</td>
<td>T</td>
<td>LABELTABLE</td>
<td>C</td>
</tr>
<tr>
<td>11</td>
<td>STAGE2</td>
<td>M</td>
<td>STAGE9</td>
<td>M</td>
</tr>
<tr>
<td>12</td>
<td>DIAL2</td>
<td>T</td>
<td>DIAL9</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>RELATIONDICT</td>
<td>C</td>
<td>STAGE10</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>STAGE3</td>
<td>M</td>
<td>DIAL10</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td>DIAL3</td>
<td>T</td>
<td>DETAILREC</td>
<td>M</td>
</tr>
<tr>
<td>16</td>
<td>DOMAIN</td>
<td>C</td>
<td>WRITEPARA</td>
<td>M</td>
</tr>
<tr>
<td>17</td>
<td>KEYDICT</td>
<td>C</td>
<td>PAGEDETAIL</td>
<td>C</td>
</tr>
<tr>
<td>18</td>
<td>SUBMODEL</td>
<td>C</td>
<td>STAGE11</td>
<td>M</td>
</tr>
<tr>
<td>19</td>
<td>STAGE3-1</td>
<td>M</td>
<td>DIAL11</td>
<td>T</td>
</tr>
<tr>
<td>20</td>
<td>STAGE4</td>
<td>M</td>
<td>STAGE12</td>
<td>M</td>
</tr>
<tr>
<td>21</td>
<td>DIAL4</td>
<td>T</td>
<td>DIAL12</td>
<td>T</td>
</tr>
<tr>
<td>22</td>
<td>DOMAINLIST</td>
<td>C</td>
<td>TITLEREC</td>
<td>M</td>
</tr>
<tr>
<td>23</td>
<td>STAGE5</td>
<td>M</td>
<td>TITLEPARA</td>
<td>M</td>
</tr>
<tr>
<td>24</td>
<td>DIAL5</td>
<td>T</td>
<td>PAGETITLE</td>
<td>C</td>
</tr>
<tr>
<td>25</td>
<td>DOMAININDEX</td>
<td>C</td>
<td>STAGE13</td>
<td>M</td>
</tr>
<tr>
<td>26</td>
<td>STAGE5-1</td>
<td>M</td>
<td>DIAL13</td>
<td>T</td>
</tr>
<tr>
<td>27</td>
<td>STAGE5-2</td>
<td>M</td>
<td>PHEADREC</td>
<td>N</td>
</tr>
<tr>
<td>28</td>
<td>HELP</td>
<td>M</td>
<td>PHEADPARA</td>
<td>M</td>
</tr>
<tr>
<td>29</td>
<td>DIALHELP</td>
<td>T</td>
<td>PAGEHEAD</td>
<td>C</td>
</tr>
<tr>
<td>30</td>
<td>WORK</td>
<td>S</td>
<td>WORK</td>
<td>S</td>
</tr>
</tbody>
</table>

C = Common data, M = Macro definition, T = Text, S = 'Sink'
Table 30.1 continued

<table>
<thead>
<tr>
<th>Mainfile name</th>
<th>GWAC132-DAT4</th>
<th>GWAC132-DAT5</th>
<th>GWAC132-MODL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Subfile Type</td>
<td>Subfile Type</td>
<td>Subfile Type</td>
</tr>
<tr>
<td>1</td>
<td>STAGE16</td>
<td>OWNRETR</td>
<td>MODEL1</td>
</tr>
<tr>
<td>2</td>
<td>DIAL16</td>
<td>OWNITL</td>
<td>MODEL2</td>
</tr>
<tr>
<td>3</td>
<td>TOTLREC</td>
<td>OWNSBK1</td>
<td>MODEL3</td>
</tr>
<tr>
<td>4</td>
<td>TOTLPARA</td>
<td>OWNSBK2</td>
<td>MODEL4</td>
</tr>
<tr>
<td>5</td>
<td>PAGETOTL</td>
<td>OWNSBK3</td>
<td>MODEL5</td>
</tr>
<tr>
<td>6</td>
<td>STAGE17</td>
<td>OWNSBK4</td>
<td>MODEL6</td>
</tr>
<tr>
<td>7</td>
<td>DIAL17</td>
<td>OWNSBK5</td>
<td>MODEL7</td>
</tr>
<tr>
<td>8</td>
<td>STAGE17-1</td>
<td>OWNSBK6</td>
<td>MODEL8</td>
</tr>
<tr>
<td>9</td>
<td>STAGE17-2</td>
<td>OWNSBK7</td>
<td>MODEL9</td>
</tr>
<tr>
<td>10</td>
<td>STAGE17-3</td>
<td>OWNSBK8</td>
<td>MODEL10</td>
</tr>
<tr>
<td>11</td>
<td>STAGE17-4</td>
<td>OWNPAGE</td>
<td>MODEL11</td>
</tr>
<tr>
<td>12</td>
<td>STAGE17-5</td>
<td>OWNSBH1</td>
<td>MODEL12</td>
</tr>
<tr>
<td>13</td>
<td>STAGE18</td>
<td>OWNSBH2</td>
<td>MODEL13</td>
</tr>
<tr>
<td>14</td>
<td>DIAL8</td>
<td>OWNSBH3</td>
<td>MODEL14</td>
</tr>
<tr>
<td>15</td>
<td>COMMENTS</td>
<td>OWNSBH4</td>
<td>MODEL15</td>
</tr>
<tr>
<td>16</td>
<td>CONSTANTS</td>
<td>OWNSBH5</td>
<td>MODEL16</td>
</tr>
<tr>
<td>17</td>
<td>COBOLPROG</td>
<td>OWNSBH6</td>
<td>MODEL17</td>
</tr>
<tr>
<td>18</td>
<td>TUPLE</td>
<td>OWNSBH7</td>
<td>MODEL18</td>
</tr>
<tr>
<td>19</td>
<td>PICTUREDICT</td>
<td>OWNSBH8</td>
<td>WORK</td>
</tr>
<tr>
<td>20</td>
<td>OWNSTART</td>
<td>OWNSTL1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>WORK</td>
<td>OWNSTL2</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>OWNSTL3</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>OWNSTL4</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>OWNSTL5</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>OWNSTL6</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>OWNSTL7</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>OWNSTL8</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>OWNTOTL</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>OWNDETAII</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>WORK</td>
<td></td>
</tr>
</tbody>
</table>

C = Common data, M = Macro definition, T = Text, S = 'Sink'
Table 30.2 Initialisation of 'grown' macro definition subfiles

<table>
<thead>
<tr>
<th>'Grown' macros requiring initialisation</th>
<th>Initial contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLEPARA STLPARA4 OWNSBK6</td>
<td>Empty macro definition consisting of the following two statements:</td>
</tr>
<tr>
<td>TITLEREC STLPARA5 OWNSBK7</td>
<td>%DEF macro-name</td>
</tr>
<tr>
<td>PHEADPARA STLPARA6 OWNSBK8</td>
<td>%END</td>
</tr>
<tr>
<td>PHEADREC STLPARA7 OWNPAGE</td>
<td></td>
</tr>
<tr>
<td>SBHPARA1 STLPARA8 OWNSBH1</td>
<td></td>
</tr>
<tr>
<td>SBHPARA2 STLREC1 OWNSBH2</td>
<td></td>
</tr>
<tr>
<td>SBHPARA3 STLREC2 OWNSBH3</td>
<td></td>
</tr>
<tr>
<td>SBHPARA4 STLREC3 OWNSBH4</td>
<td></td>
</tr>
<tr>
<td>SBHPARA5 STLREC4 OWNSBH5</td>
<td></td>
</tr>
<tr>
<td>SBHPARA6 STLREC5 OWNSBH6</td>
<td></td>
</tr>
<tr>
<td>SBHPARA7 STLREC6 OWNSBH7</td>
<td></td>
</tr>
<tr>
<td>SBHPARA8 STLREC7 OWNSBH8</td>
<td></td>
</tr>
<tr>
<td>SBHREC1 STLREC8 OWNDETAIL</td>
<td></td>
</tr>
<tr>
<td>SBHREC2 TOTLPARA OWNSTL1</td>
<td></td>
</tr>
<tr>
<td>SBHREC3 TOTLREC OWNSTL2</td>
<td></td>
</tr>
<tr>
<td>SBHREC4 OWNSTART OWNSTL3</td>
<td></td>
</tr>
<tr>
<td>SBHREC5 OWNRETR OWNSTL4</td>
<td></td>
</tr>
<tr>
<td>SBHREC6 OWNTITL OWNSTL5</td>
<td></td>
</tr>
<tr>
<td>SBHREC7 OWNSBK1 OWNSTL6</td>
<td></td>
</tr>
<tr>
<td>SBHREC8 OWNSBK2 OWNSTL7</td>
<td></td>
</tr>
<tr>
<td>STLPARA1 OWNSBK3 OWNSTL8</td>
<td></td>
</tr>
<tr>
<td>STLPARA2 OWNSBK4 OWNTOTL</td>
<td></td>
</tr>
<tr>
<td>STLPARA3 OWNSBK5</td>
<td></td>
</tr>
</tbody>
</table>
Table 30.3 Initialisation of common data subfiles

<table>
<thead>
<tr>
<th>Common data subfile name</th>
<th>Stage first used</th>
<th>Initial contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIONDICT</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DOMAINDICT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SUBMODEL</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>DOMAINLIST</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>DOMAININDEX</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>PAGEDETAIL</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>PAGETITLE</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>PAGEHEAD</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>PAGESBH1</td>
<td>14</td>
<td>All subfiles have one record which contains a zero character.</td>
</tr>
<tr>
<td>PAGESBH2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESBH3</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESBH4</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESBH5</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESBH6</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESBH7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESBH8</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PAGESTL1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGESTL8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PAGETOTL</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
# APPENDIX VIII

**COBOL REPORT PROGRAM GENERATION DETAILS**

<table>
<thead>
<tr>
<th>Section Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Steering Lines</td>
<td>3</td>
</tr>
<tr>
<td>2. Identification Division</td>
<td>3</td>
</tr>
<tr>
<td>3. Environment Division</td>
<td>3</td>
</tr>
<tr>
<td>4. Data Division Preliminaries</td>
<td>4</td>
</tr>
<tr>
<td>5. File Section</td>
<td>6</td>
</tr>
<tr>
<td>6. Working-Storage Section Level 77 Entries</td>
<td>8</td>
</tr>
<tr>
<td>7. Group Description Entries</td>
<td>11</td>
</tr>
<tr>
<td>8. Procedure Division Entries</td>
<td>15</td>
</tr>
<tr>
<td>9. Z-Initial Section, Paragraph Z-PARA-1</td>
<td>17</td>
</tr>
<tr>
<td>10. Paragraph Z-PARA-2</td>
<td>18</td>
</tr>
<tr>
<td>11. Z-Processing Section, Paragraph Z-PARA-3</td>
<td>19</td>
</tr>
<tr>
<td>12. Paragraph Z-PARA-4</td>
<td>20</td>
</tr>
<tr>
<td>13. Paragraph Z-PARA-5</td>
<td>21</td>
</tr>
<tr>
<td>14. Z-Final Section, Paragraph Z-PARA-FINAL</td>
<td>21</td>
</tr>
<tr>
<td>15. Paragraph Z-Close-Down</td>
<td>23</td>
</tr>
<tr>
<td>16. Z-Other-Procedures Section</td>
<td>23</td>
</tr>
<tr>
<td>17. Paragraph Z-PARA-ADD</td>
<td>24</td>
</tr>
<tr>
<td>18. Paragraph Z-PARA-WRITE</td>
<td>24</td>
</tr>
<tr>
<td>19. Paragraphs Z-End-Page-Counts and Z-Start-Line</td>
<td>25</td>
</tr>
<tr>
<td>20. Paragraphs Z-HeadI and Z-HeadI-Exit</td>
<td>26</td>
</tr>
<tr>
<td>21. Paragraphs Z-TOTALI</td>
<td>28</td>
</tr>
<tr>
<td>22. Paragraph Z-PARA-TITLE</td>
<td>29</td>
</tr>
<tr>
<td>23. Paragraph Z-PARA-PAGE</td>
<td>29</td>
</tr>
<tr>
<td>24. Paragraph Z-HeadI-Write</td>
<td>31</td>
</tr>
<tr>
<td>25. Paragraphs Z-TOTALI-Write</td>
<td>32</td>
</tr>
<tr>
<td>26. Paragraph Z-TOTALO-WRITE</td>
<td>33</td>
</tr>
<tr>
<td>27. Z-ReadI Section</td>
<td>33</td>
</tr>
<tr>
<td>28. Z-Sequence-Break Section</td>
<td>34</td>
</tr>
</tbody>
</table>
29. Z-FETCH-TUPLE SECTION 38
30. PARAGRAPH Z-END-OF-DATA-TEST 38
31. PARAGRAPH Z-SET-TUPLE-KEY 39
32. PARAGRAPH Z-SET-TUPLE-DATA-1 42
33. PARAGRAPH Z-OPTION-1 47
34. END OF PROGRAM 48
ILLUSTRATIONS

Figure 4.1 Identification and processing of data items 6
Figure 6.1 Use of level 77 data entries 10
Figure 7.1 'Grown' macros containing report line formats 14
Figure 8.1 Generated COBOL program outline flowchart 16
Figure 20.1 Outline processing for sequence break headings 27
Figure 28.1 Outline processing for the Z-SEQUENCE-BREAK section 35
Figure 31.1 Outline processing for setting Tuple keys 40
Figure 32.1 Outline processing for setting up tuple data from a master file 43
Figure 32.2 Outline processing for setting up tuple data from a non-master file 44
The text in this appendix describes the generation of a COBOL report program in the general case where the data is on more than one input file and is sequenced on more than one key. The description is presented in program listing order:

1. Steering lines
2. Identification division
3. Environment division
4. Data division preliminaries
5. File section
6. Working-storage section level 77 entries
7. Group description entries
8. Procedure division entries
9. Z-INITIAL section, paragraph Z-PARA-1
11. Z-INITIAL section, paragraph Z-PARA-3
12. Paragraph Z-PARA-4
13. Paragraph Z-PARA-5
14. Z-FINAL section, paragraph Z-PARA-FINAL
15. Paragraph Z-CLOSE-DOWN
16. Z-OTHER-PROCEDURES section
17. Paragraph Z-PARA-ADD
18. Paragraph Z-PARA-WRITE
19. Paragraphs Z-END-PAGE-COUNTS and Z-START-LINE
20. Paragraphs Z-HEADi and Z-HEADi-EXIT
21. Paragraphs Z-TOTALi
22. Paragraph Z-PARA-TITLE
23. Paragraph Z-PARA-PAGE
24. Paragraph Z-HEADi-WRITE
25. Paragraphs Z-TOTALi-WRITE
26. Paragraph Z-TOTALO-WRITE

27. Z-READ-i section

28. Z-SEQUENCE-BREAK section

29. Z-FETCH-TUPLE section

30. Paragraph Z-END-OF-DATA-TEST

31. Paragraph Z-SET-TUPLE-KEY

32. Paragraph Z-SET-TUPLE-DATA-1

33. Paragraph Z-OPTION-i

34. End of program
1. STEERING LINES

The steering lines are input parameters for the COBOL compiler which specify the options required. They are independent of the user's program statements and form a separate input file. For the installation where the practical work was developed it was appropriate to generate the following statements:

*IDENTITY COBOL
*CALLEE
*CALLEE CARDS (GENPROG)
*OBJECT EDS (ICLA-DEFAULT)
*STANDARD
*CONSOLIDATE EDS XPCK
*SUBGROUPS EDS (SUBGROUPS-RS)
*LOAD
*LIST S.P
****

The above code is easily varied to suit other circumstances.

2. IDENTIFICATION DIVISION

The following statements are generated for this division:

IDENTIFICATION DIVISION.
PROGRAM-ID. COBOL.
DATE-WRITTEN, today's-date.

where today's-date is the value in system global variable #CDT which is set by the DATE macro-time statement.

3. ENVIRONMENT DIVISION

The general form of the statements generated for this division is as follows:

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ICL-1907.
OBJECT-COMPUTER. ICL-1907 MEMORY SIZE 19000 WORDS.
SPECIAL-NAMES.
PRINTER IS TEST-PRINTER
*DATE IS Z-DATE *TIME IS Z-TIME.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT Z-REPORT-FILE ASSIGN TO PRINTER 1.
SELECT Z-FILE-1 ASSIGN TO hardware-name integer.

SELECT Z-FILE-n ASSIGN TO hardware-name integer.
The source and object computer statements are tailored to the installation where the practical work was developed. The memory size was set to be the maximum permitted for running programs on-line; larger programs had to be run in batch mode. The above code is easily varied to suit a different environment.

In the SPECIAL-NAME paragraph the test-printer statement is only generated if the user opted to have a message displayed when a mis-match file condition is detected, i.e. global variable $GMOP is equal to 'Y' (Appendix VII Section 14). The DATE and TIME statements are a necessary preliminary which enables Procedure division statements to access the date and time values in the operating system executive at program execution time.

The input files are those described in the SUBMODEL subfile (Section 7.10 of main text) and, for the purpose of program generation, are named in the order in which they occur as Z-FILE-1, Z-FILE-2, etc up to Z-FILE-n, where n is the value in global variable $NREL. The hardware-name for each file is derived from the File medium field in each Type 3 File name record in the SUBMODEL subfile. The following shows the derivation:

<table>
<thead>
<tr>
<th>SUBMODEL</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>File medium</td>
<td>Hardware-name</td>
</tr>
<tr>
<td>CR</td>
<td>CARD-READER</td>
</tr>
<tr>
<td>ED</td>
<td>EDS</td>
</tr>
<tr>
<td>MT</td>
<td>TAPES</td>
</tr>
<tr>
<td>PT</td>
<td>PAPER-READER</td>
</tr>
</tbody>
</table>

The integer value is associated with the hardware-name. It starts with 1 and is incremented by 1 for each file with the same hardware-name.

4. DATA DIVISION PRELIMINARIES

The name of each input data item occurs in at least three group items within the Data division: the record description in the File section and the TUPLE and TUPLE-N groups in the Working-
storage section. Although in the case of numeric data items the
USAGE clause will differ, the PICTURE clause will be the same. It
is therefore convenient to record in the PICTUREDICT subfile a
dictionary of data item names and their corresponding PICTURE. The
format of the PICTUREDICT subfile is described in tabular form in
Section 10 of Appendix V.

The data from which the PICTURE is derived, i.e. length,
type and decimal point location, is located in the fields of the
DOMAININDEX subfile records (Appendix V Section 7). The
DOMAININDEX subfile is read record by record, the required data
extracted and the PICTURE string generated. The corresponding
PICTUREDICT subfile record may then be inserted. DOMAININDEX
contains details of user created temporary items as well as those
from the input files. For the temporary data items empty picture
strings are inserted in the corresponding PICTUREDICT subfile
record. This ensures a one to one correspondence between the
records of the DOMAININDEX and PICTUREDICT subfiles.

During the pass through the records of the DOMAININDEX
subfile, the statements for the TUPLE and SAVETOTL macro
definitions are also generated and filed as they too are derived
from the same data. These macros are called during the generation
of the Working-storage section (Section 7 of this appendix).

The body of the TUPLE macro contains the data description
statements for all input file items required to solve the user's
problem. The macro has the following form:

```
%DEF TUPLE
%   LABELOFF
02 data-name PICTURE picture-string USAGE COMPUTATIONAL.
   LABELON
%END
```

The USAGE COMPUTATIONAL clause is present only in numeric data
descriptions.
The body of the SAVETOTL macro contains data description statements for all totalling items, both input and temporary. The general form of the SAVETOTL macro is as follows:

```
%DEF SAVETOTL
% LABELOFF
  03 data-name PICTURE picture-string
    VALUE 0 USAGE COMPUTATIONAL.
  03 -----
    -----
% LABELON
%END
```

The numeric item picture-strings for use in the PICTUREDICT subfile and TUPLE macro are generated using logic similar to that shown in Figure 12.2 of Appendix VII. The picture-string used for a totalling item in the SAVETOTL subfile is similarly generated, but the field length is always taken to be 18. This is the maximum size permitted by ICL COBOL for items used in calculations. The picture-string used for alphanumeric data items in both the PICTUREDICT subfile and the TUPLE macro is always X(n), where n is the length of the field as contained in the DOMAININDEX record for the item.

Figure 4.1 summarises the identification and processing for the various data item types described in the DOMAININDEX subfile.

Figure 4.1 Identification and processing of data items

<table>
<thead>
<tr>
<th>DOMAININDEX subfile record</th>
<th>PICTUREDICT picture-string generated</th>
<th>TUPLE macro data description generated</th>
<th>SAVETOTL macro data description generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>T</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓ empty string used</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>S</td>
<td>✓ empty string used</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
5. FILE SECTION

The File section, the first of the Data division, contains the file and record description statements for all the files used by the program. The statements generated to identify the division and section and those for the output report file are generated without reference to any common data subfiles. However, in order to generate the statements for the input file descriptions use must be made of the data in the SUBMODEL and PICTUREDICT subfiles.

The first six statements generated for the Data division are as follows:

```
DATA DIVISION.
FILE SECTION.
FD Z-REPORT-FILE
  LABEL RECORDS OMITTED
  DATA RECORD IS Z-OUTREC.
  01 Z-OUTREC PICTURE X(n).
```

where \( n \) is the page width in terms of the number of characters which can be accomodated. If global variable \#WIDTH is equal to 'N' or blank then \( n \) has the value 70, otherwise \( n \) has the value 120.

The input files have already been designated Z-FILE-1, Z-FILE-2, etc in the order in which their details are recorded in the SUBMODEL subfile (Section 3 of this appendix). The following shows the form of the statements for each of these input file descriptions:

```
FD Z-FILE-i
  BLOCK CONTAINS block-size CHARACTERS
  RECORD CONTAINS record-size CHARACTERS
  LABEL RECORD STANDARD VALUE OF ID "file-label"
  DATA RECORD IS Z-INREC-i.
```

where

- \( i \) is the \( i \)th file described in the SUBMODEL common data file
- block-size, record-size and file-label are as in the 2nd, 3rd and 4th fields of the \( i \)th File name record type 3 of the SUBMODEL subfile.
For character peripherals, i.e. card readers and paper tape readers, the LABEL RECORDS OMITTED clause replaces that shown above.

The file description entries for each user file are generated at three levels using the data in both the SUBMODEL and PICTUREDICT subfiles. The 01 level is the record name as previously designated in the FD statement, i.e.

01 Z-INREC{i.

where i is the ith data file described in SUBMODEL.

For each input file described in SUBMODEL only those field-names which have a matching record in the PICTUREDICT subfile (Section 4 of this appendix) have a level 03 data description generated. This takes the form:

03 field-name PICTURE picture-string.

A binary search of the PICTUREDICT subfile is made to determine if a matching field-name record exists. If so, the picture-string is extracted from the PICTUREDICT record and used to generate the above COBOL statement.

In addition to the field description statements level 03 FILLER statements are generated when one of the following conditions occurs:

1. The first field in the description does not start in the first position of the record.

2. Two successive fields are not contiguous.

3. The last field of the description does not end in the last position of the record.

The start position, length of field and the record size details from the SUBMODEL File and Field description records are used to identify the above conditions. The FILLER statements generated have the following form:

03 FILLER PICTURE X(n),
where n is the number of character positions between fields.

The level 02 record description statements are necessary to cope with the cases where SUBMODEL field descriptions are not entered in start position order or when fields overlap. The first level 02 statement for a record description is as follows:

```
02 Z-IN-RECi-1.
```

where i is the ith file described in SUBMODEL. Subsequent level 02 statements are generated when either of the conditions mentioned above occurs and takes the form:

```
02 Z-IN-RECi-k,REDEFINES Z-IN-RECi-j.
```

where k = j+1 and Z-IN-RECi-j is the name of the previous level 02 statement.

The overall structure of a typical record description is as shown below:

```
01 Z-INREC1.
  02 Z-IN-REC1-1.
    03 FILLER PICTURE ....
    03 data-name-1 PICTURE ....
    03 data-name-2 PICTURE ....
    03 FILLER PICTURE ....
  02 Z-IN-REC1-2 REDEFINES Z-IN-REC1-1.
    03 FILLER PICTURE ....
    03 data-name-3 PICTURE ....
    03 FILLER PICTURE ....
    03 data-name-4 PICTURE ....
    03 FILLER PICTURE ....
```

6. WORKING-STORAGE SECTION LEVEL 77 ENTRIES

The following statements, which define the section and various non-contiguous data items required by the Procedure division of the program, are generated. The use of each variable is outlined in Figure 6.1
where

\( n \) is equal to the number of input files as contained in global variable \( \#NREL \)

and

\( l \) is the maximum number of lines per report page as specified in global variable \( \#GMXLN \).
Figure 6.1 Use of level 77 data entries

<table>
<thead>
<tr>
<th>COBOL data-name</th>
<th>Program usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z-EOF-1</td>
<td>End of file indicators; one for each input file.</td>
</tr>
<tr>
<td>Z-EOF-2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Z-EOF-n</td>
<td></td>
</tr>
<tr>
<td>Z-EOF-COUNT</td>
<td>End of file counter; used to determine if all the input files have been read to the end.</td>
</tr>
<tr>
<td>Z</td>
<td>Used as subscripts in the calculations which determine the last line on which any sequence break heading or detail print group may start.</td>
</tr>
<tr>
<td>Z1</td>
<td></td>
</tr>
<tr>
<td>Z-LINE-COUNT</td>
<td>Line count variable.</td>
</tr>
<tr>
<td>Z-PAGE-SWITCH</td>
<td>Program switch used to indicate that a page heading and sequence break headings, if required, have just been output.</td>
</tr>
<tr>
<td>Z-ADV-LINES</td>
<td>Variable used for the ADVANCING option of the first WRITE statement of certain print groups. It is used to carry forward the count of blank lines after the last non-blank line of the previous print group.</td>
</tr>
<tr>
<td>Z-IGNORE</td>
<td>Used as a program switch to show that a mis-match file condition has been detected and the 'Ignore' option has been invoked.</td>
</tr>
<tr>
<td>Z-MAX-LINES</td>
<td>Maximum number of lines per report page as specified in global variable #GMXLN.</td>
</tr>
</tbody>
</table>
7. GROUP DESCRIPTION ENTRIES

Depending on the user's problem group descriptions may be generated for the following groups:

1. Z-CONTROL-TOTALS
2. Z-LINE-COUNTS
3. Z-CONTROL-BREAK
4. Report line formats
5. Z-TUPLE
6. Z-TUPLE-N

7.1 Z-CONTROL-TOTALS

The Z-CONTROL-TOTALS group description is generated only if the user requested the totalling facility, i.e. global variable #GTOT is equal to 1.

A level 02 statement is generated for the final totals and for each intermediate sequence key level except the minor key. The level 02 items are named Z-LEVEL-i where i is the sequence key level number or 0 in the case of final totals. Thus if n is the number of sequence keys in the user's data, as stored in global variable #NKEY, then i takes the values n-1, n-2, ..., 1, 0.

For each level 02 statement a nested call is made to the SAVETOTL macro (Section 4 of this appendix) which re-generates the level 03 statements for each totalling item.

The following macro-time statements show how the Z-CONTROL-TOTALS group description may be generated:

```
% IF #GTOT.EQ.0,G0T 31
% #LV1=#NKEY
C1 Z-CONTROL-TOTALS.
%30 #LV1=#LV1-1
O2 Z-LEVEL-#LV1.
[SAVETOTL]
% IF #LV1.NE.0,G0T 30
%31 CONTINUE
```

For the model problem described in Section 5.9 of the main text the following statements would be generated:
7.2 Z-LINE-COUNTS

The Z-LINE-COUNTS group contains a data description entry for each sequence break heading, if specified, in major to penultimate minor key order and the detail print group. The data in the Z-LINE-COUNTS group is used by the Procedure division statements to calculate the lowest line of a report page on which the printing of a particular print group may begin. The contents of the group description is redefined so that subscripts may be used in the Procedure division to reference the data items.

The level 02 entries are called Z-L-C-1, Z-L-C-2, etc and each has an initial value assigned. The initial value is equal to the number of lines in the print group represented. The general form of the statements generated for this group description is as follows:

```
01 Z-LINE-COUNTS.
02 Z-L-C-1 PICTURE 99 VALUE \( l_1 \) USAGE COMPUTATIONAL.
02 Z-L-C-2 PICTURE 99 VALUE \( l_2 \) USAGE COMPUTATIONAL.
    \vdots
02 Z-L-C-\( p \) PICTURE 99 VALUE \( l_p \) USAGE COMPUTATIONAL.
01 FILLER REDEFINES Z-LINE-COUNTS.
02 Z-L-C PICTURE 99 OCCURS \( p \) TIMES USAGE COMPUTATIONAL.
```

where \( p \) is one plus the number of sequence break headings used in the report. The \( l_1, l_2, \ldots, l_p \) values are extracted from the first record of the PAGESBHn subfiles in the case of sequence break headings and the PAGEDETAIL subfile for the detail line print group. The value in global variable \$GHEAD indicates whether or
not sequence break headings have been specified by the user.

7.3 Z-CONTROL-BREAK

The Z-CONTROL-BREAK group is used by the Procedure division statements to detect a sequence break in the input data in any key except the minor key. The group contains a level 02 data description for each key in major to penultimate minor key order. The number of keys in the user's data is contained in global variable $\#NKEY and the names of the keys are stored in the KEYDICT subfile (Appendix V Section 5). The picture-string for each key is extracted from the corresponding key-name record in the PICTUREDICT subfile (Appendix V Section 10). The general form of the statements generated for the Z-CONTROL-BREAK group is as follows:

```
01 Z-CONTROL-BREAK.
  02 major-key-name PICTURE picture-string.
  ;
  02 penultimate-minor-key-name PICTURE picture-string.
```

7.4 REPORT LINE FORMATS

The group descriptions for the report line formats of the various categories of output specified by the user were generated during the problem specifying dialogue. They are located in the bodies of the macros with REC as the macro name suffix which were 'grown' during the various stages of the dialogue. The group descriptions are re-generated by making nested calls to these 'grown' macros. Since all the macros were initialised as empty, the macro for any option not invoked by the user will be empty and no statements will be generated. Figure 7.1 shows the names of the macros containing group descriptions for report lines together with a note of the problem specifying stage in which they were 'grown'. 
### Figure 7.1 'Grown' macros containing report line formats

<table>
<thead>
<tr>
<th>Macro name</th>
<th>Stage 'grown'</th>
<th>Main text reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLEREC</td>
<td>12</td>
<td>7.20</td>
</tr>
<tr>
<td>PHEADREC</td>
<td>13</td>
<td>7.21</td>
</tr>
<tr>
<td>SBHREC1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBHREC8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DETAILREC</td>
<td>10</td>
<td>7.18</td>
</tr>
<tr>
<td>STLREC1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STLREC8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTLREC</td>
<td>16</td>
<td>7.24</td>
</tr>
</tbody>
</table>

### 7.5 Z-TUPLE

The Z-TUPLE group contains data descriptions for all the items which are used to solve the user's problem. The group is made up of three parts:

1. **The additional data items** - date, time and page number.
2. **The items in the user's problem domains**.
3. **User created temporary items**.

The statements for parts 2 and 3 have already been generated and they are located in the bodies of the TUPLE and TEMPITEM macros (Section 4 of this appendix and Section 7.12 of the main text). These statements are re-generated by making nested calls.
to the macros. The statements generated for the first part of the
Z-TUPLE group are always the same and are shown below:

```
01 Z-TUPLE.  
  02 ZDATE PICTURE X(8).  
  02 ZTIME PICTURE X(8).  
  02 Z-PAGE-COUNT PICTURE 9999 VALUE 0 USAGE COMPUTATIONAL.  
```

7.6 Z-TUPLE-N

The Z-TUPLE-N group contains descriptions of all the data
items from the user's files which are required to solve the problem.
(c.f. part 2 of Z-TUPLE.) In the Procedure division, when the
input file records have been matched, the input data items are
temporarily stored in Z-TUPLE-N. After the sequence break checking
has been carried out the Z-TUPLE-N items are moved to the
corresponding items in the Z-TUPLE group. The level 02 data
description statements for the Z-TUPLE-N group are generated by
making a nested call to the TUPLE macro.

8. PROCEDURE DIVISION ENTRIES

The Procedure division processing is based on three main
sections, Z-INITIAL, Z-PROCESSING and Z-FINAL (Figure 8.1). In
addition there are other sections whose execution is initiated by
means of PERFORM statements. Each section consists of several
paragraphs. The statements generated for the various sections and
paragraphs of the Procedure division are discussed in the
subsequent sections of this appendix.

In the general forms of generated statements, which are
listed in the subsequent section of this appendix, positions
where statements from 'grown' macros are incorporated are
indicated by enclosing the macro name in square brackets.
Figure 8.1 Generated COBOL program outline flowchart

**Z-INITIAL SECTION**

1. **START**
   - **Z-PARA-1**
     - OPEN FILES

2. **OPEN FILES**
   - **Z-PARA-2**
     - PROCESS LINE COUNTERS
     - ACCEPT DATE & TIME
     - READ ALL INPUT FILES

3. ** FETCH TUPLE PROCESSING**
   - AT END
     - IGNORE TUPLE
     - SET UP FIRST TUPLE
     - SET INITIAL SEQUENCE BREAK KEY VALUES
     - TITLE PROCESSING
     - PAGE PROCESSING

4. **WRITE REPORT DATA**
   - **Z-PARA-4**
     - ACCUMULATE TOTALS
     - SELECT TUPLE

5. ** FETCH TUPLE PROCESSING**
   - AT END
     - IGNORE TUPLE
     - SEQUENCE BREAK PROCESSING

**Z-PROCESSING SECTION**

6. **CONCATENATE DATA ITEMS**
   - **Z-PARA-3**
     - OWN CODE PROCESSING

7. **FINAL SECTION**
   - **Z-PARA-5**
     - WRITE REPORT DATA

8. **FINAL PROCESSING**
   - **Z-CLOSEDOWN**
     - STOP RUN
9. Z-INITIAL SECTION, PARAGRAPH Z-PARA-1

The processing carried out by the statements generated for the Z-PARA-1 paragraph of the Z-INITIAL section consists of the following:

1. Open all files used by the program.

2. Execute, by means of PERFORM statements using the VARYING option, the Z-END-PAGE-COUNTS and Z-START-LINE procedures which are located in the Z-OTHER-PROCEDURES section. These procedures calculate and store in the Z-L-C data items the last line number on which certain print groups may begin to print.

3. Obtain the run time and date values in character form from the operating system executive by means of ACCEPT operations. Use is made of data names previously defined in the SPECIAL-NAMES paragraph.

4. Execute the own code processing, if any, to be carried out at the start of the processing run. The COBOL statements for this processing step were generated during Stage 18 of the problem specifying dialogue (Section 7.26 of the main text) and are re-generated by making a nested call to the OWNSTART macro.

5. Read the first record from each of the input files by means of PERFORM statements which refer to Sections Z-READ-1, Z-READ-2, etc which are described in Section 27 of this appendix.

The general form of the statements generated for the above processing is shown below:

```
DATA DIVISION.
Z-INITIAL SECTION.
Z-PARA-1. OPEN INPUT
    Z-FILE-1
    Z-FILE-2
    ...
    Z-FILE-n
OUTPUT Z-REPORT-FILE.
PERFORM Z-END-PAGE-COUNTS VARYING Z FROM p BY -1 UNTIL Z
    EQUALS 1.
PERFORM Z-START-LINE VARYING Z FROM 1 BY 1 UNTIL Z GREATER
    THAN p.
ACCEPT ZDATE IN Z-TUPLE FROM Z-DATE.
```
ACCEPT ZTIME IN Z-TUPLE FROM Z-TIME.

[OWNSTART]
PERFORM Z-READ-1.
PERFORM Z-READ-2.

PERFORM Z-READ-n.

where

n is the number of input files as contained in global variable ^GREL

and

p is equal to 1 if sequence break headings were not specified, i.e. ^GHEAD = 0, and equal to ^GKEY if ^GHEAD is equal to 1.

10. PARAGRAPH Z-PARA-2

The processing carried out by the statements generated for the Z-PARA-2 paragraph is as follows:

1. Set up the input data for the first tuple in Z-TUPLE-N by performing the Z-FETCH-TUPLE section, whose processing is described in Section 29 of this appendix.

2. Test the Z-IGNORE switch set in Z-FETCH-TUPLE. If it is equal to 1 a file mismatch condition exists which the user wishes to ignore, so processing resumes at step 1 above.

3. Move the input data for the first tuple into the Z-TUPLE group.

4. Set the initial values for the key items in the Z-CONTROL-BREAK group equal to the corresponding items of the first tuple. The Z-CONTROL-BREAK group is later used to detect a sequence break in the input data keys.

5. Execute the own code processing to be carried out prior to writing the report title. The COBOL statements for this step are in the body of the OWNTITL macro which was 'grown' during the
problem specifying dialogue (Section 7.18 of main text). They are regenerated by making a nested call to the OWNTITL macro.

6. Write out the Report title, if specified, by performing the Z-PARA-TITLE paragraph of the Z-OTHER-PROCEDURES section.
N.B. The statements for steps 5 and 6 are only generated if global variable #GTITL is equal to 1.

7. Skip to the top of a new report page and write out the page heading etc, if specified. This processing is carried out by performing the Z-PARA-PAGE paragraph in the Z-OTHER-PROCEDURES section.

The statements generated for the above processing are as follows:

Z-PARA-2.
PERFORM Z-FETCH-TUPLE. IF Z-IGNORE EQUALS 1 GO TO Z-PARA-2.
MOVE CORRESPONDING Z-TUPLE-N TO Z-TUPLE.

[OWNRETR]
MOVE CORRESPONDING Z-TUPLE TO Z-CONTROL-BREAK.

[OWNTITL]
PERFORM Z-PARA-TITLE.
PERFORM Z-PARA-PAGE.

11. Z-PROCESSING SECTION, PARAGRAPH Z-PARA-3

In the Z-PARA-3 paragraph the following three tasks may be carried out:

1. Execute the MOVE operations which concatenate data into the compound temporary items, if any were created by the user (Section 7.12 of main text). The COBOL statements, if any, for this processing are located in the body of the CONCAT macro and may be regenerated by making a nested call to the macro.

2. Execute the own code processing, if any, to be carried out after a data retrieval. The COBOL statements for this processing task were generated during Stage 18 of the problem
specifying dialogue (Section 7.26 of main text). They are regenerated by making a nested call to the OWNRETR macro.

3. Apply the data selection conditions, if any, specified during the problem specifying dialogue. If the conditions are satisfied control passes to paragraph Z-PARA-4, otherwise it passes to Z-PARA-5. The COBOL statements for this task are located in the body of the SELCOND macro (Section 7.14 of main text) and are regenerated by making a nested call to the macro.

The statements generated for the above processing are as follows:

Z-PROCESSING SECTION.
Z-PARA-3.

[CONCAT]
[OWNRETR]
[SELCOND]

12. PARAGRAPH Z-PARA-4

One or two tasks are performed in the Z-PARA-4 paragraph depending on whether or not the user requested the totalling facility:

1. If totalling was requested the Z-PARA-ADD paragraph, which accumulates the required data values, is executed by means of a PERFORM statement. This statement is only generated if global variable #TOT is equal to 1.

2. The Z-PARA-WRITE paragraph is performed so that the detail line(s) for the current data retrieval are written on the Report file.

The Z-PARA-ADD and Z-PARA-WRITE paragraphs are located in the Z-OTHER-PROCEDURES section and are described later.

The statements generated for the Z-PARA-4 paragraph are as follows:
13. PARAGRAPH Z-PARA-5

The processing carried out by the statements generated for the Z-PARA-5 paragraph is as follows:

1. Set up the input data for the next tuple in the Z-TUPLE-N group by performing the Z-FETCH-TUPLE section.

2. Test the Z-IGNORE switch set in the Z-FETCH-TUPLE section. If it is equal to 1 the user wishes to ignore a file mismatch condition, so processing resumes at step 1 above.

3. Execute the Z-SEQUENCE-BREAK section by means of a PERFORM statement. This section detects and processes an input data sequence break condition and is described later.

4. Pass control to the Z-PARA-3 paragraph where the next tuple is processed.

The statements generated for the Z-PARA-5 paragraph are shown below and are always the same as they are independent of the user's problem:

```
Z-PARA-5. PERFORM Z-FETCH-TUPLE.
IF Z-IGNORE EQUALS 1 GO TO Z-PARA-5.
PERFORM Z-SEQUENCE-BREAK. GO TO Z-PARA-3.
```

14. Z-FINAL SECTION, PARAGRAPH Z-PARA-FINAL

If totalling was not specified by the user, i.e. global variable $TQT is equal to zero, the Z-PARA-FINAL paragraph is a null paragraph and only the following statements are generated for it:

```
Z-FINAL SECTION.
Z-PARA-FINAL. EXIT.
```

If, however, the totalling facility was invoked the following processing is carried out in the Z-PARA-FINAL paragraph:
1. Process the sequence break subtotals in penultimate minor to major key order. This processing is initiated by PERFORM statements which refer to the Z-TOTALi paragraphs, where i is the sequence key level. These paragraphs are in the Z-OTHER-PROCEDURES section and are described later.

2. Test the value of the line counter to see if there is enough room at the bottom of the page for all the lines of the final totals print group. If there is insufficient space take a new page by performing the page heading paragraph Z-PARA-PAGE.

3. Obey the user's own code processing, if any, specified for execution prior to printing the final totals. The COBOL statements for this own code processing were generated during the problem specifying dialogue (Section 7.26 of main text) and are located in the body of the OWN TOTL macro. They are regenerated by making a nested call to the OWN TOTL macro.

4. Output the totals in the user specified format by performing the Z-TOTALO-WRITE paragraph located in the Z-OTHER-PROCEDURES section.

COBOL statements for steps 2 to 4 are generated only if totals output was specified by the user, i.e. global variable is #GTOTL is equal to 1.

If the totalling facility was used and totals output is required the general form of the statements generated for the Z-PARA-FINAL paragraph is as shown below:

```
Z-FINAL SECTION.
Z-PARA-FINAL.
   PERFORM Z-TOTALi.
   | | | 
   PERFORM Z-TOTAL1.
   IF Z-LINE-COUNT IS GREATER THAN 1 PERFORM Z-PARA-PAGE.
   [OWN TOTL]
   PERFORM Z-TOTALO-WRITE.
```

where

i takes the values n-1, n-2, ..., 1 and n is the number of
sequence key levels as contained in global variable #GNKEY

and

1. is the value in global variable #GMXLN minus the value in
   the control record of the PAGETOTL work subfile (Appendix V
   Section 9).

15. PARAGRAPH Z-CLOSE-DOWN

The Z-CLOSE-DOWN paragraph carries out two tasks:

1. Close all the input files and the output report file.

2. Stop the program run.

The statements generated for this paragraph have the
following general form:

Z-CLOSE-DOWN. CLOSE
   Z-FILE-1
   Z-FILE-2
   ...
   Z-FILE-n
   Z-REPORT-FILE. STOP RUN.

where n is the number of input files as contained in global
variable #GNREL.

16. Z-OTHER-PROCEDURES SECTION

The Z-OTHER-PROCEDURES section is a convenient means of

- grouping a number of paragraphs which do not fit into any of
  the other sections of the Procedure division. The execution of
  each paragraph is initiated by means of a PERFORM statement. With
  the exception of the Z-TOTALi, Z-HEADi and Z-HEADi-EXIT paragraphs,
  the ordering of paragraphs within the section is quite arbitrary.

Depending on the user's problem the statements for some paragraphs
may not be generated.

The following statement is generated to define the section:

Z-OTHER-PROCEDURES SECTION.
The Z-PARA-ADD paragraph is only generated if the totalling facility was invoked by the user, i.e. global variable GTOT is equal to 1.

The processing in this paragraph causes the totalling items from the current tuple to be added to the corresponding items at the penultimate minor key level of the Z-CONTROL-TOTALS group.

If generated, the statement for the Z-PARA-ADD paragraph is as follows:

```
Z-PARA-ADD. ADD CORRESPONDING Z-TUPLE TO Z-LEVEL-i.
```

where i is equal to n-1 and n is the value in global variable #ONKEY.

The processing in the Z-PARA-WRITE paragraph consists of the following:

1. Test the line counter to see if there is enough room on the current report page to output the detail line(s) for the current tuple. If there is not enough space a new page is taken by performing the page heading paragraph Z-PARA-PAGE.

2. Carry out the own code processing, if any, specified by the user during the problem specifying dialogue for execution prior to printing the detail line(s) for the current tuple (Section 7.26 of the main text). The COBOL statements for this step are located in the body of the OWNDETAIL macro and are regenerated by making a nested call to it.

3. Output the detail line(s) for the current tuple in the format specified during the problem dialogue (Section 7.18 of main text). The body of the WRITEPARA macro contains the COBOL statements for this step and they may be regenerated by making a
nested call to the macro.

The statements generated for the Z-PARA-WRITE paragraph have the following general form:

Z-PARA-WRITE.
  IF Z-LINE-COUNT GREATER THAN Z-L-C(1) PERFORM Z-PARA-PAGE.
  [OWNDETAIL]
  [WRITEPARA]

where 1 is equal to 1 if sequence break headings were not specified, i.e. global variable $\#$HEAD is equal to zero, and equal to $\#$NKEY if $\#$HEAD is equal to 1.

19. PARAGRAPHS Z-END-PAGE-COUNTS AND Z-START-LINE

The calculations for determining the last line of a report page on which any sequence break heading or detail line print group may begin are delayed until the COBOL program is executed. This is because the necessary calculations are cumbersome to effect using the PG/2 macro processor facilities which do not support subscripts and array variables.

These calculations aim to avoid sequence break headings appearing at the foot of a report page and to prevent incomplete print groups from having to be continued on the next report page. A sequence break heading is not output to a report page unless there is also room for all the sequence break headings minor to it and at least one set of detail output.

The execution of the statements in the Z-END-PAGE-COUNTS and Z-START-LINE paragraphs is initiated by PERFORM statements in the Z-PARA-1 paragraph of the Z-INITIAL section (Section 9 of this appendix). Use is made of the data items in the Z-LINE-COUNTS group which are referenced by means of subscript variables Z and Z1.
The following statements are generated for these paragraphs:

\[
\begin{align*}
\text{Z-END-PAGE-COUNTS. COMPUTE } Z1 &= Z - 1, \\
\text{ADD } Z-L-C(Z) \text{ TO } Z-L-C(Z1), \\
\text{Z-START-LINE. COMPUTE } Z-L-C(Z) &= Z-\text{MAX-LINES} - Z-L-C(Z).
\end{align*}
\]

20. PARAGRAPHS Z-HEADi AND Z-HEADi-EXIT

Paragraphs Z-HEADi and Z-HEADi-EXIT control the writing out of the sequence break heading at the ith key level. Statements for these paragraphs are only generated if the user specified sequence break headings, i.e. global variable \#GHEAD is equal to 1 (Section 7.22 of main text). A pair of paragraphs is generated for each sequence break heading in the user's report, i.e. i takes the values 1 to n-1 where n is number of sequence keys as in global variable \#NKEY. The execution of these paragraphs is initiated by a PERFORM ... THRU ... statement in the Z-SEQUENCE-BREAK section.

The processing carried out by each pair of paragraphs consists of the following steps:

1. Test the value of Z-LINE-COUNT to see if there is enough room on the report page to print the sequence break heading. If there is insufficient space continue at step 4, otherwise continue below.

2. Test the state of the page switch Z-PAGE-SWITCH. If it is equal to 1 a new page has just been taken so control passes to step 6 as there is no need to repeat the sequence break heading.

3. Output the sequence break heading for the ith key level by performing the Z-HEADi-WRITE paragraph. Continue processing at step 5.

4. Skip to a new page and output the headings by performing the Z-PARA-PAGE paragraph.

5. Exit from the range of the PERFORM statement.

The above processing is illustrated in Figure 20.1 and the
form of the COBOL statements generated for these pairs of paragraphs is shown below:

Z-HEAdi.
  IF Z-LINE-COUNT GREATER THAN Z-L-C(i) PERFORM Z-PARA-PAGE
  GO TO Z-HEAdi-EXIT.
  IF Z-PAGE-SWITCH EQUALS 1 GO TO Z-HEAdi-EXIT.
  PERFORM Z-HEAdi-WRITE.
Z-HEAdi-EXIT. EXIT.

where i is equal to the sequence break key level which takes the values 1 to n-1, where n is the value in global variable $NKEY.

Figure 20.1 Outline processing for sequence break headings
21. PARAGRAPHS Z-TOTALi

Paragraph Z-TOTALi controls the summation and writing out of the sequence break subtotals at key level i. The statements for these paragraphs are generated only if the totalling facility was invoked, i.e. if global variable #GTOT is equal to 1.

COBOL statements for steps 1 to 4 of the processing carried out in this paragraph and described below are generated only if sequence break subtotal output was specified by the user, i.e. global variable #SUBT is equal to 1 (Section 7.23 of main text).

1. Test the line counter Z-LINE-COUNT to see if there is room for the subtotals print group on the current page. If there is insufficient space take a new page by performing the Z-PARA-PAGE paragraph.

2. If own code processing was specified for execution prior to printing the subtotals at the current key level carry out this processing. The COBOL statements for this own code processing were generated during the problem specifying dialogue and are located in the body of the OWNSTLi macro, where i is the current key level. These COBOL statements are regenerated by making a nested call to the OWNSTLi macro.

3. Output the subtotals for the current key level by performing the Z-TOTALi-WRITE paragraph.

4. Turn the page switch off by setting Z-PAGE-SWITCH equal to zero. This indicates that a non-heading print group has just been output to the report page.

5. Add the subtotals of items at the current key level to the corresponding items of the next key level major to it.

6. Zeroise the subtotals of items at the current key level.

The general form of the COBOL statements generated for these paragraphs is show below:
Z-TOTALi.
   IF Z-LINE-COUNT GREATER THAN 1 PERFORM Z-PARA-PAGE.

[OWNSTLi]

PERFORM Z-TOTALi-WRITE.
MOVE 0 TO Z-PAGE-SWITCH.
ADD CORRESPONDING Z-LEVEL-i TO Z-LEVEL-j.
SUBTRACT CORRESPONDING Z-LEVEL-i FROM Z-LEVEL-i.

where

i  is the current key level which takes the values n-1 to 1,
   where n is the number of sequence keys as in global variable $NKEY.

j  is equal to i-1,

and

l  is equal to the value in global variable $GMXLN minus the
   value in the control record of the PAGESTLi work subfile.
   (Appendix VII Section 16 and Appendix V Section 9).

22. PARAGRAPH Z-PARA-TITLE

The Z-PARA-TITLE paragraph is only present in the generated
COBOL program if the user specified a Report title during the
problem dialogue (Section 7.20 of the main text). The COBOL
statements for this paragraph were generated at the time of the
dialogue and are located in the body of the TITLEPARA macro.
They may be regenerated by making a nested call to the TITLEPARA
macro.

23. PARAGRAPH Z-PARA-PAGE

The Z-PARA-PAGE paragraph controls the skip to the top of
the next page of the report, the incrementing of the page counter
and the output of headings, if specified.

The processing steps are as follows:
1. Carry out the own code processing, if any, specified
during the problem dialogue for execution prior to the output of a
page heading. The COBOL statements for this processing were
generated at the time of the dialogue and are located in the body
of the OWNPAGE macro (Appendix VII Section 26). They may be
regenerated by making a nested call to the macro.

2. Clear the output file record area Z-OUTREC to spaces.
3. Increment the page counter Z-PAGE-COUNT by 1.
4. Skip to the top of a new report page by writing out a
blank line on the report and then skipping to channel 1.
5. Output the page heading, if specified by the user during
the problem dialogue (Section 7.21 in main text). The COBOL
statements for this step are located in the body of the PHEADPARA
macro and may be regenerated by making a nested call to the macro.
6. Write out the sequence break headings for all key levels
in major to penultimate minor key order by performing the
Z-HEADi-WRITE paragraphs, where i denotes the key level. The
COBOL statements for this step are only generated if sequence
break headings were specified by the user, i.e. if global
variable $HEAD is equal to 1.

The general form of the COBOL statements generated for the
Z-PARA-PAGE paragraph is as follows:

Z-PARA-PAGE.

[OWNPAGE]

MOVE SPACES TO Z-OUTREC. ADD 1 TO Z-PAGE-COUNT IN Z-TUPLE.
WRITE Z-OUTREC BEFORE ADVANCING CHANNEL-1.

[PHEADPARA]

PERFORM Z-HEAD1-WRITE.

PERFORM Z-HEADi-WRITE.

where i is equal to the value in global variable $GNKEY minus 1.
24. PARAGRAPHS Z-HEADi-WRITE

The Z-HEADi-WRITE paragraph writes out the lines of the sequence break heading for the ith key level. If sequence break headings were specified during the problem dialogue, i.e. global variable #GHEAD is equal to 1, a paragraph is generated for each key level except the minor key.

The tasks carried out by the COBOL statements generated for each Z-HEADi-WRITE paragraph are as follows:

1. The own code processing, if any, specified during the problem dialogue for execution prior to the output of the sequence break heading at the current key level, is carried out. The COBOL statements for this task are located in the OWNSBHI macro subfile and may be regenerated by making a nested call to the macro (Appendix VII Section 26).

2. The lines of the sequence break heading for the current key level are set up and written out. The SBHPARAi macro contains the COBOL statements for this step which were generated during the problem dialogue (Section 7.22 of main text). They may be regenerated by making a nested call to the SBHPARAi macro (Appendix VII Section 22).

The general form of the statements generated for the Z-HEADi-WRITE paragraphs is as follows:

Z-HEADi-WRITE.

[OWNSBHI]

[SBHPARAi]

where i is the sequence key level and takes the values 1, 2, ..., n-1 and n is the value in global variable #GKEY.
25. **PARAGRAPHS Z-TOTALi-WRITE**

These paragraphs are generated only if the totalling facility was invoked and sequence break subtotals were specified during the problem dialogue, i.e. global variables GTOT and GSUBT are both equal to 1. A Z-TOTALi-WRITE paragraph is generated for each key level i except the minor key.

Each Z-TOTALi-WRITE paragraph carries out two tasks, the COBOL statements for which were generated during the problem specifying dialogue:

1. Carry out the own code processing, if any, to be executed prior to writing the current sequence break subtotals. The COBOL statements for this task are located in the body of the OWNSTLi macro and are regenerated by making a nested call to the macro (Appendix VII Section 26).

2. Write out the lines of the sequence break subtotals print group for the current key level in the format specified by the user. The STLPARAi macro contains the COBOL statements for this task, where i is the current key level. The statements may be regenerated by making a nested call to the macro (Appendix VII Section 23).

The general form of the statements generated for the Z-TOTALi-WRITE paragraph is as follows:

```
Z-TOTALi-WRITE.
  [OWNSTLi]
  [STLPARAi]
```

where i is the sequence key level and takes the values 1, 2, ..., n-1 and n is the value in global variable GKEY.
26. PARAGRAPH Z-TOTALO_WRITE

The Z-TOTALO_WRITE paragraph is present in the generated program only if the user invoked the totalling facility and specified totals output during the problem dialogue, i.e. global variables #GTQT and #GTOTL are both equal to 1 (Section 7.24 of main text).

The Z-TOTALO_WRITE paragraph carries out two tasks, the COBOL statements for which were generated during the problem specifying dialogue:

1. Carry out the own code processing, if any, to be executed prior to writing the final totals. The COBOL statements for this task are located in the body of the OWNTOTL macro and are regenerated by making a nested call to the macro (Appendix VII Section 26).

2. Write out the lines of the final totals print group. The TOTLPara macro contains the statements for this task and they may be regenerated by making a nested call to the macro (Appendix VII Section 24).

The form of the statements generated for this paragraph is as follows:

Z-TOTALO_WRITE.
[OWNTOTL]
[TOTLPara]

27. Z-READ-i SECTION

For each input file used by the COBOL program a Z-READ-i section is generated. The files are numbered in the order in which they are described in the SUBMODEL work subfile. Thus i takes the values 1 to n, where n is the value in global variable #GNREL.
Input files whose number of sequence keys is equal to the value in global variable $\texttt{NKEY}$ are, for the purposes of the COBOL generating system, designated 'Master' files. Master files are processed in a different way from non-master files. In order to determine whether or not a file is a master file it is necessary to inspect the eighth field of the File name type 3 record in the SUBMODEL subfile (Appendix V Section 3). A count of the number of master files described in SUBMODEL is maintained in global variable $\texttt{NKEY}$. This value is later used in the generation of the Z-FETCH-TUPLE section.

In the processing carried out in each of these sections a record is read from the input file. If the end of file is detected the end of file indicator (Appendix VIII Section 6) is set equal to 1. If the file is a master file the value of Z-EOF-COUNT is incremented by 1 when the end is reached.

The execution of these sections is initiated by means of PERFORM statements within other sections of the program. The general form of the statements generated for each section is shown below:

\begin{verbatim}
Z-READ-i SECTION.
Z-READ-i-1. READ Z-FILE-i AT END GO TO Z-READ-i-2.
    GO TO Z-READ-i-EXIT.
Z-READ-i-2. MOVE 1 TO Z-EOF-i.
    ADD 1 TO Z-EOF-COUNT.
Z-READ-i-EXIT. EXIT.
\end{verbatim}

where \( i \) is the file number which takes the values 1 to \( n \) and \( n \) is the value in global variable $\texttt{NREL}$.

N.B. The ADD statement in the third paragraph is generated only if the file is a master file.

28. Z-SEQUENCE-BREAK SECTION

The processing in the Z-SEQUENCE-BREAK section detects a sequence break in any key level except the minor and controls the output of sequence break subtotals and headings, if specified.
Figure 28.1 Outline processing for the Z-SEQUENCE-BREAK section
The section consists of pairs of paragraphs, one pair for each key level except the minor key. These paragraphs are called Z-BREAK-i and Z-BREAK-i-EXIT, where i denotes the key level. The section concludes with two other paragraphs, Z-BREAK-MOVE and Z-BREAK-EXIT. The general structure of this section is illustrated in Figure 28.1.

The processing within the pairs of Z-BREAK-i and Z-BREAK-i-EXIT paragraphs is as follows:

1. Test to see if there is a change in the value of the key at the current level, i.e. compare its value in the Z-TUPLE-N and Z-CONTROL-BREAK groups. If the values are the same continue processing at step 7, otherwise continue below.

2. If own code processing was specified for execution at a sequence break in the current key level, carry out this processing. The COBOL statements for this own code processing were generated during the problem specifying dialogue and are located in the OWNSBHI macro, where i is the current key level. These COBOL statements are regenerated by making a nested call to the OWNSBHI macro (Appendix VII Section 26).

3. Carry out the sequence break subtotals processing for the current key level and all levels minor to it. These are executed in minor to major key sequence. The COBOL statements for this step are generated only if the totalling facility has been invoked, i.e. global variable $GTOT is equal to 1.

4. Move the input data items from the Z-TUPLE-N group to the Z-TUPLE group.

5. Update the values of the keys in the Z-CONTROL-BREAK group by setting them equal to those in the Z-TUPLE group.

6. Output the sequence break headings for the current key level and all keys minor to it. The COBOL statements for this step are generated only if sequence break headings were specified.
by the user during the problem dialogue, i.e. global variable $GHEAD$ is equal to 1 (Section 7.22 of main text).

7. Exit from the processing at the current key level.

The Z-BREAK-MOVE paragraph is executed when there is no sequence break in the input data. It causes the data from the Z-TUPLE-N group to be moved to the Z-TUPLE group read for processing.

The Z-BREAK-EXIT paragraph consists only of an EXIT statement and provides a common end to the processing paths within the section.

The general form of the COBOL statements generated for the Z-SEQUENCE-BREAK section is as follows. The KEYDICT common data subfile (Appendix V Section 5) is referenced in order to relate the sequence key level to the corresponding key name.

```
Z-SEQUENCE-BREAK SECTION.
Z-BREAK-i.
    IF key-i IN Z-TUPLE-N EQUALS key-i IN Z-CONTROL-BREAK GO TO Z-BREAK-i-EXIT.

    [OWNSBHI]
    PERFORM Z-TOTALj THRU Z-TOTALi.
    MOVE CORRESPONDING Z-TUPLE-N TO Z-TUPLE.
    MOVE CORRESPONDING Z-TUPLE TO Z-CONTROL-BREAK.
    PERFORM Z-HEADj THRU Z-HEADj-EXIT.
    GO TO Z-BREAK-EXIT.
Z-BREAK-i-EXIT. EXIT.

Z-BREAK-MOVE. MOVE CORRESPONDING Z-TUPLE-N TO Z-TUPLE.
Z-BREAK-EXIT. EXIT.
```

where

- $i$ is the sequence break key level which takes the values 1 to $n-1$, where $n$ is the value in global variable $GNKEY$

and

- $j$ is equal to $n-1$, where $n$ is the value in global variable $GNKEY$.

key-i is the name of the ith sequence key as found in the ith record of the KEYDICT subfile.
N.B. When \( n \) is equal to 2 the THRU option is omitted from the PERFORM Z-TOTAL statement.

29. Z-FETCH-TUPLE SECTION

The processing carried out in this section matches the records from the various input files and from them selects the data for the current tuple in the Z-TUPLE-N group. It also detects when all the input data has been processed and handles the file mismatch condition according to the option selected by the user during the problem dialogue (Section 7.13 of main text).

The three main tasks within the section are listed below together with the name of the paragraph at which the task processing begins:

2. Set tuple keys (Z-SET-TUPLE-KEY).
3. Set up non-key tuple data and process file mismatch condition (Z-SET-TUPLE-DATA-1).

The COBOL statements generated for processing each input file depend on whether or not the file is a Master file (Section 27 of this appendix). In order to generate the statements for this section reference is made to the data in the SUBMODEL, KEYDICT and DOMAININDEX common data subfiles.

The following statement is generated to define the section:

```
Z-FETCH-TUPLE SECTION.
```

30. PARAGRAPH Z-END-OF-DATA-TEST

When all the records on all the master files have been read and processed the end of data is deemed to have been reached. This condition is detected by testing the Z-EOF-COUNT variable which is incremented by 1 each time the end of a master file is
reached (Section 27 of this appendix). If Z-EOF-COUNT is equal to the value in global variable #GMSTR then control is passed to the Z-FINAL section. If the end of data has not been reached the Z-IGNORE switch is turned off by moving a zero to it.

The COBOL statements generated for this paragraph are as follows:

```
Z-END-OF-DATA-TEST. IF Z-EOF-COUNT EQUALS m GO TO Z-FINAL.
MOVE O TO Z-IGNORE.
```

where m is the value in global value #GMSTR, i.e. the number of master files used by the program.

31. PARAGRAPH Z-SET-TUPLE-KEY

In this paragraph the sequence keys of all the master files are compared in order to find the file with the lowest key. The key values from this file are moved to the key items in the Z-TUPLE-N group.

The flowchart in Figure 31.1 shows the outline processing for the above task when more than one master file is used by the generated program. If only one master file is used the flowchart can be simplified so that only the block marked with an asterisk is retained.

In the simplified case with only one master file the following shows the general form of the COBOL statements generated for the Z-SET-TUPLE-KEY paragraph:

```
Z-SET-TUPLE-KEY.
MOVE key-1 IN Z-INRECx TO key-1 IN Z-TUPLE-N.
MOVE key-2 IN Z-INRECx TO key-2 IN Z-TUPLE-N.
: : :
MOVE key-n IN Z-INRECx TO key-n IN Z-TUPLE-N.
```

where the master file is the xth file described in the SUBMODEL common data subfile.

Key-1, key-2, ..., key-n are the key names from the KEYDICT subfile
Figure 31.1 Outline processing for setting the Tuple keys

First Master file

ENTER

2-SET-TUPLE-KEY

END OF Z-FILE-X

N

Y

2-SET-HIGH-KEY

MOVE Z-FILE-X KEYS TO Z-TUPLE-N KEYS

Move 9’s to numeric keys in 2-TUPLE-N
Move high values to alphanumeric keys in 2-TUPLE-N

Z-KEYS-Y

N

Y

Repeat for subsequent Master files

END OF Z-FILE-Y

2-FILE-Y KEYS < Z-TUPLE-N KEYS

2-MOVE-Y

MOVE Z-FILE-Y KEYS TO Z-TUPLE-N KEYS

EXIT
and \( n \) is the number of sequence keys as contained in global variable \( ^N\text{SNKEY} \).

When more than one master file is used the general form of the COBOL statements generated for setting up the tuple key is as follows:

\[
\text{Z-SET-TUPLE-KEY,}
\]

\[
\begin{align*}
\text{IF } & \text{Z-EOF-x EQUALS 1 GO TO Z-SET-HIGH-KEY.} \\
\text{MOVE } & \text{key-1 IN Z-INRECx TO key-1 IN Z-TUPLE-N.} \\
\text{MOVE } & \text{key-2 IN Z-INRECx TO key-2 IN Z-TUPLE-N.} \\
\text{MOVE } & \text{key-n IN Z-INRECx TO key-n IN Z-TUPLE-N.} \\
\end{align*}
\]

First master file

\[
\text{GO TO Z-SET-CONTINUE.}
\]

\[
\text{Z-SET-HIGH-KEY.}
\]

\[
\begin{align*}
\text{MOVE } & \text{high-constant TO key-1 IN Z-TUPLE-N.} \\
\text{MOVE } & \text{high-constant TO key-2 IN Z-TUPLE-N.} \\
\text{MOVE } & \text{high-constant TO key-n IN Z-TUPLE-N.}
\end{align*}
\]

\[
\text{Z-SET-CONTINUE. EXIT.}
\]

\[
\text{Z-KEYS-y. IF } \text{Z-EOF-y EQUALS 1 GO TO Z-KEYS-y-EXIT.}
\]

\[
\begin{align*}
\text{IF } & \text{key-1 IN Z-INRECy GREATER THAN key-1 IN Z-TUPLE-N GO TO Z-KEYS-y-EXIT.} \\
\text{IF } & \text{key-1 IN Z-INRECy LESS THAN key-1 IN Z-TUPLE-N GO TO Z-MOVE-y-1.} \\
\text{IF } & \text{key-2 IN Z-INRECy GREATER THAN key-2 IN Z-TUPLE-N GO TO Z-KEYS-y-EXIT.} \\
\text{IF } & \text{key-2 IN Z-INRECy LESS THAN key-2 IN Z-TUPLE-N GO TO Z-MOVE-y-2.}
\end{align*}
\]

Other master files

\[
\begin{align*}
\text{IF } & \text{key-n IN Z-INRECy GREATER THAN key-n IN Z-TUPLE-N GO TO Z-KEYS-y-EXIT.} \\
\text{IF } & \text{key-n IN Z-INRECy LESS THAN key-n IN Z-TUPLE-N GO TO Z-MOVE-y-n.} \\
\end{align*}
\]

\[
\text{GO TO Z-SET-KEYS-y-EXIT.}
\]

\[
\text{Z-MOVE-y-1. MOVE key-1 IN Z-INRECy TO key-1 IN Z-TUPLE-N.}
\]

\[
\text{Z-MOVE-y-2. MOVE key-2 IN Z-INRECy TO key-2 IN Z-TUPLE-N.}
\]

\[
\text{Z-MOVE-y-n. MOVE key-n IN Z-INRECy TO key-n IN Z-TUPLE-N.}
\]

\[
\text{Z-KEYS-y-EXIT. EXIT.}
\]

where

\( x \)

denotes the number of the first master file described in the SUBMODEL common data subfile.
y denotes the number of a subsequent master file described in the SUBMODEL subfile.

key-1, key-2, ..., key-n are the key names from the KEYDICT subfile and n is the number of sequence keys as contained in global variable $\#GNKEY. The keys are in major to minor key order.

high-constant is the figurative constant HIGH-VALUES when the key is of alphanumeric type. For a numeric type key it is a numeric literal such that all digit positions of the tuple key field will be filled with 9's. The type, length and decimal point location of the key field is described in its DOMAINDEX subfile record.

32. PARAGRAPH Z-SET-TUPLE-DATA-1.

In the Z-SET-TUPLE-DATA-1 and subsequent paragraphs the keys in the input file records are matched against the keys in the Z-TUPLE-N group. If a file record with matching keys exists the non-key items are moved from the record to the corresponding items in the Z-TUPLE-N group. If, for any file, a matching key record cannot be found, the error option selected by the user is carried out (Section 7.13 of main text).

During the search for matching records more than one pass through a non-master file may be required, but for master files only one pass is made. The files are processed in the order in which they are described in the SUBMODEL subfile. The processing of master and non-master files differs and the details are illustrated in Figures 32.1 and 32.2. When only one master file is required to solve the problem, the flowchart can be simplified so that only the blocks marked with an asterisk are retained.
Figure 32.1 Outline processing for setting up tuple data from a meter file.
Figure 32.2 Outline processing for setting up tuple data from a non-master file

ENTRY

Z-SET-TUPLE-DATA-PROC

END OF Z-FILE-PROC

N

Z-MATCH-PROC

FILE KEY EQUALS TUPLE KEY

FILE KEY > TUPLE KEY

Y

Z-FETCH-PROC

Z-READ-PROC

READ A RECORD FROM Z-FILE-PROC

END OF Z-FILE-PROC

N

CLOSE Z-FILE-PROC AND OPEN Z-FILE-PROC

TURN OFF END OF FILE INDICATOR, i.e. MOVE 0 TO Z-EOF-PROC

Z-READ-PROC

READ A RECORD FROM Z-FILE-PROC

Z-OPTION-PROC

MISMATCH PROCESSING

Z-MOVE-PROC-EXIT

EXIT

MOVE CORRESPONDING NON-KEY ITEMS FROM FILE RECORD TO Z-TUPLE-PROC


The general form of the COBOL statements generated to process a master file is as follows:

```
  IF Z-EOF-p EQ UALS 1 GO TO Z-OPTION-p.
  IF key-1 IN Z-INRECp NOT EQUAL TO
    key-1 IN Z-TUPLE-N GO TO Z-OPTION-p.
  IF key-2 IN Z-INRECp NOT EQUAL TO
    key-2 IN Z-TUPLE-N GO TO Z-OPTION-p.
    ...
  IF key-n IN Z-INRECp NOT EQUAL TO
    key-n IN Z-TUPLE-N GO TO Z-OPTION-p.
  MOVE item-a IN Z-INRECp TO item-a
    IN Z-TUPLE-N.
  MOVE item-b IN Z-INRECp TO item-b
    IN Z-TUPLE-N.
  PERFORM Z-READ-p.
  GO TO Z-MOVE-p-EXIT.
Z-OPTION-p.
```

(See Section 33 of this appendix for the code generated.)

```
Z-MOVE-p-EXIT. EXIT.
```

where

- `p` is the number of the master file description in the SUBMODEL subfile.
- `key-1, key-2, ..., key-n` are the key names from the KEYDICT subfile
- `n` is the number of sequence keys as contained in global variable `#GNKEY`. The keys are in major to minor key order.
- `item-a, item-b, etc` are the names of the non-key fields in the master file record which are required to solve the user's problem, i.e. only those non-key fields whose names appear in both the SUBMODEL and DOMAININDEX subfiles.

When the user's problem uses only one master file the COBOL coding generated simplifies to the following form:

```
  MOVE item-a IN Z-INRECp TO item-a
    IN Z-TUPLE-N.
  MOVE item-b IN Z-INRECp TO item-b
    IN Z-TUPLE-N.
  PERFORM Z-READ-p.
```
For a non-master file the general form of the COBOL statements generated is as follows:

```
Z-SET-TUPLE-DATA-q.
   IF Z-EOF-q EQUALS 1 GO TO Z-OPTION-q.
Z-MATCH-q.
   IF key-r IN Z-INRECq GREATER THAN key-r IN Z-TUPLE-N GO TO Z-OPTION-q.
   IF key-r IN Z-INRECq LESS THAN key-r IN Z-TUPLE-N GO TO Z-FETCH-q.
   IF key-s IN Z-INRECq GREATER THAN key-s IN Z-TUPLE-N GO TO Z-OPTION-q.
   IF key-s IN Z-INRECq LESS THAN key-r IN Z-TUPLE-N GO TO Z-FETCH-q.

MOVE item-s IN Z-INRECq TO item-s IN Z-TUPLE-N.
MOVE item-b IN Z-INRECq TO item-b IN Z-TUPLE-N.
GO TO Z-MOVE-q-EXIT.
Z-FETCH-q. PERFORM Z-READ-q.
   IF Z-EOF-q EQUALS 0 GO TO Z-MATCH-q.
   CLOSE Z-FILE-q RETAIN OPEN INPUT Z-FILE-q MOVE 0 TO Z-EOF-q.
   PERFORM Z-READ-q.
Z-OPTION-q.

(See Section 33 of this appendix for the code generated.)
```

Z-MOVE-q-EXIT. EXIT.

where

- **q** is the number of the non-master file description in the **SUBMODEL** subfile.
- **key-r, key-s, etc** are the names of the key fields used to sequence the non-master file records. The keys are in major to minor key order.
- **item-s, item-b, etc** are the names of the non-key fields in the non-master file record which are required to solve the user's problem, i.e. only those non-key fields whose names appear in both the **SUBMODEL** and **DOMAININDEX** subfiles.
33. PARAGRAPH Z-OPTION-1

The COBOL statements generated for the Z-OPTION-i paragraphs, where i denotes the ith file described in the SUBMODEL subfile, depend on the mismatch option selected by the user during the problem specifying dialogue (Section 7.13 of main text). The selected option is indicated by the contents of global variables #GMOP and #GOPT and string variable #S05.

If global variable #GMOP is equal to 'Y' the user wishes a message to be output when a file mismatch condition is detected. In this case the following statements are generated for the Z-OPTION-i paragraph:

```
  DISPLAY "relation-name", "error-message", "KEY VALUES key-r”, key-s
```

where

- relation-name is the File-name as described in the SUBMODEL subfile,
- error-message is the text contained in string variable #S05.
- key-r, key-s, etc are the names of the keys used to sequence the file in which the mismatch condition is detected. The keys are listed in major to minor order.

The DISPLAY statement, if generated, is followed by the statements which process the mismatch condition according to the user selected option. If #GOPT is equal to 1 the user wishes to ignore the mismatch condition and continue processing the data files. In this case the following COBOL statement is generated:

```
  MOVE 1 TO Z-IGNORE.
```

If global variable #GOPT is equal to 2 the user wishes to create a dummy matching record with numeric non-key items.
containing zeros and alphanumeric non-key items containing blanks.

To implement this option the following statements are generated:

```
MOVE figurative-constant TO item-a IN Z-TUPLE-N.
MOVE figurative-constant TO item-b IN Z-TUPLE-N.
```

where

- `item-a`, `item-b`, etc are the names of the non-key fields in the
  record of the file which are required to solve the user's
  problem, i.e. only those non-key fields which appear in both
  the SUBMODEL and DOMAININDEX subfiles.

- `figurative-constant` is ZEROS for a numeric non-key field and SPACES
  for an alphanumeric non-key field. The field type is
  determined from the Field description record in the SUBMODEL
  subfile.

If `#GOPT` is equal to 3 the user wishes to terminate the
processing of the data. This option is implemented by the
generation of the following statement:

```
GO TO Z-CLOSE-DOWN.
```

34. END OF PROGRAM

The final statement generated for the COBOL program consists
of four asterisks. This statement indicates to the COBOL compiler
the end of the source program.
APPENDIX IX

DIALOGUE SAMPLES

1. STAGE 1
2. STAGE 2
3. STAGE 3
4. STAGE 4
5. STAGE 5
6. STAGE 6
7. STAGE 7
8. STAGE 8
9. STAGE 9
10. STAGE 10
11. STAGE 11
12. STAGE 12
13. STAGE 13
14. STAGE 14
15. STAGE 15
16. STAGE 16
17. STAGE 17
18. STAGE 18
19. CREATE CATALOGUE
20. CHANGE CATALOGUE PASSWORD
21. LIBRARIAN
22. LIBPRELIM AND LIBDELETE
23. LIBPRELIM AND LIBADD
APPENDIX IX

DIALOGUE SAMPLES

This appendix contains dialogue samples for the COBOL report program generation system. Sections 1 to 18 show the dialogue carried out with the casual user for the specification of the model problem described in Section 5.9 of the main text. Each stage of the problem specification is shown in a separate section. Sections 19 to 23 contain samples of the dialogue with the database administrator produced by the catalogue maintenance macros.

N.B. The type face of the teletype on which these listings were produced causes £ to appear as $ and $ to appear as \\. 
STAGE 1 - INTRODUCTION

THIS SYSTEM IS A SELF-TEACHING COBOL PROGRAM GENERATOR.

BY MEANS OF A QUESTION AND ANSWER DIALOGUE THE SYSTEM WILL ENABLE YOU TO

SPECIFY BOTH YOUR PROBLEM AND THE DATA TO BE USED. THE SYSTEM WILL THEN

GENERATE A COBOL PROGRAM TO PROVIDE YOU WITH YOUR ANSWERS.

IF AT ANY TIME DURING THE DIALOGUE YOU WISH TO CHANGE YOUR MIND ABOUT AN

EARLIER DECISION OR REQUIRE ASSISTANCE PLEASE TYPE %HELP. SHOULD YOU

WISH TO ABANDON THE SESSION PLEASE TYPE CTRL A INSTEAD.

IN MANY INSTANCES THE SYSTEM OFFERS YOU A DEFAULT RESPONSE TO A

QUESTION. TO TAKE ADVANTAGE OF THIS OPTION YOU MERELY PRESS THE ACCEPT

KEY ON YOUR TERMINAL KEYBOARD.

THE SYSTEM TAKES A RELATIONAL VIEW OF THE DATABASE.

THE DATABASE IS A COLLECTION OF STORED DATA AVAILABLE FOR USE IN THE

SOLUTION OF A PROBLEM.

THE DATABASE IS MADE UP OF DOMAINS, EACH OF WHICH CONTAINS A SET OF DATA

ITEMS RELATING TO A PARTICULAR ATTRIBUTE.

E.G. AN EMPLOYEE-NAME DOMAIN CONTAINS THE SET OF ALL EMPLOYEE NAMES AND

A DEPARTMENT-NO DOMAIN CONTAINS THE SET OF ALL DEPARTMENT NUMBERS.

A DIFFERENT NUMBER OF ITEMS MAY BE PRESENT IN EACH DOMAIN SET.

A RELATION MAY BE DEFINED ON SEVERAL DOMAINS SUCH THAT FOR EVERY ITEM

IN ONE DOMAIN OF THE RELATION THERE EXISTS ONE AND ONLY ONE

CORRESPONDING ITEM IN EACH OF THE OTHER DOMAINS IN THE RELATION.

E.G. A PERSONNEL RELATION MAY BE DEFINED ON THE DOMAINS EMPLOYEE-NAME,

ADDRESS AND DEPARTMENT-NO. THIS EACH EMPLOYEE BELONGS TO ONLY ONE

DEPARTMENT AND LIVES AT ONE ADDRESS, ALTHOUGH SEVERAL EMPLOYEES

MAY BELONG TO THE SAME DEPARTMENT OR LIVE AT THE SAME ADDRESS.

THE EASIEST WAY TO VISUALISE THE DATA IN A RELATION IS IN TABULAR FORM,

WHERE THE DOMAIN NAMES PROVIDE THE COLUMN HEADINGS AND THE ITEMS FROM

EACH DOMAIN SET PROVIDE THE ROW ENTRIES.

THIS THE DATA IN THE PERSONNEL RELATION MENTIONED ABOVE COULD BE

CONSIDERED AS A TABLE OF WHICH THE FOLLOWING IS AN EXTRACT:-

PERSONNEL RELATION

-------------

EMPLOYEE-NAME | ADDRESS | DEPARTMENT-NO

-------------

BROWN A.B. | 124 HIGH STREET, SOUTHON | 14
BROWN L.M. | 124 HIGH STREET, SOUTHON | 02
BLACK C.D.E. | 131 MILL LANE, WESTHAM | 10
JONES T. | 14 CHURCH ROAD, EASTLEY | 02
RILEY S.M. | THE CREST, HILL RISE, NORTHAM | 06
SCOTT P.J. | 17 CASTLE STREET, SOUTHON | 09

THE GENERATED COBOL PROGRAM WILL RETRIEVE YOUR DATA IN SUCH A WAY THAT

YOU MAY CONSIDER THE DATA IN THE RELATION TABLE TO BE AVAILABLE ON A

ROW BY ROW BASIS, ONE ROW AT A TIME.
STAGE 2 - PASSWORD DIALOGUE
ACCESS TO THE SYSTEM DATABASE IS BY MEANS OF A RELATION NAME AND ITS ASSOCIATED PASSWORD. THE RELATION NAMES AND PASSWORDS ARE ALLOCATED BY THE DATABASE ADMINISTRATOR TO WHOM AN APPLICATION TO USE THE SYSTEM SHOULD BE MADE. DATA FROM MORE THAN ONE RELATION MAY BE REQUESTED TO SOLVE YOUR PROBLEM.
PLEASE TYPE THE NAME OF THE FIRST RELATION WHOSE DATA YOU WISH TO ACCESS
*:PERSONNEL
PLEASE TYPE THE PASSWORD FOR RELATION PERSONNEL
*:PERS3
HAVE YOU ANY MORE RELATIONS TO ENTER? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
*:Y
PLEASE TYPE THE NAME OF THE NEXT RELATION WHOSE DATA YOU WISH TO ACCESS
*:STAFF
PLEASE TYPE THE PASSWORD FOR RELATION STAFF
*:STAFFA
HAVE YOU ANY MORE RELATIONS TO ENTER? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
*:Y
PLEASE TYPE THE NAME OF THE NEXT RELATION WHOSE DATA YOU WISH TO ACCESS
*:DEPARTMENT
PLEASE TYPE THE PASSWORD FOR RELATION DEPARTMENT
*:DEPT4
HAVE YOU ANY MORE RELATIONS TO ENTER? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
*:N
STAGE 3 - USERS DATABASE SUBMODEL

The domains in each relation of your database will be shown below together with details of the size and type of the items they contain. The size indicates the number of characters in each item of the domain and the type shows if they are numeric (N) or alphanumeric (A). The decimal point of a numeric item is assumed to be after the right-most character unless otherwise indicated. The point location shows the direction, left (L) or right (R), together with a character count relative to the right-most character.

E.G. ITEM SIZE POINT LOCATION ITEM CONTENTS NUMERIC VALUE OF CONTENTS

5 L2 17463 174.63
3 R3 468 468000.

Where data from a relation is readily available in an ordered sequence, the key domains which determine that sequence will also be listed in order, starting with the major key (key no. 1) and finishing with the minor key.

RELATION NAME DEPARTMENT
DETAILS OF DOMAINS WITHIN THE RELATION
DOMAIN NAME ITEM SIZE TYPE POINT LOCATION
DEPT-NO 2 N
DEPT-NAME 20 A
DEPT-LOC 20 A
DEPT-MANAGER 20 A
KEY NO. KEY DOMAIN NAME
1 DEPT-NO

RELATION NAME PERSONNEL
DETAILS OF DOMAINS IN THE RELATION
DOMAIN NAME ITEM SIZE TYPE POINT LOCATION
EMPLOYEE-NO 4 N
ADDRESS 40 A
SEX 1 A
MARITAL-ST 1 A
JOIN-YEAR 2 N
JOIN-MONTH 2 N
JOIN-DAY 2 N
KEY NO. KEY DOMAIN NAME
1 EMPLOYEE-NO

RELATION NAME STAFF
DETAILS OF DOMAINS WITHIN THE RELATION
DOMAIN NAME ITEM SIZE TYPE POINT LOCATION
DEPT-NO 2 N
EMPLOYEE-NO 4 N
EMPLOYEE-NAME 20 A
KEY NO. DOMAIN NAME
1 DEPT-NO
2 EMPLOYEE-NO
4. STAGE 4

STAGE 4 - SELECTION OF RELEVANT DOMAINS
YOUR DATA BASE CONTAINS A NUMBER OF DOMAINS NOT ALL OF WHICH MAY BE
REQUIRED TO SOLVE YOUR CURRENT PROBLEM. THE SYSTEM IS ABOUT TO LIST THE
CONTENTS OF YOUR DATABASE DOMAIN BY DOMAIN. AFTER EACH DOMAIN YOU
WILL BE ASKED WHETHER OR NOT THE CURRENT DOMAIN FEATURES IN YOUR
PROBLEM. SHOULD YOU SO DESIRE, THE SYSTEM WILL AUTOMATICALLY ACCUMULATE A TOTAL
FOR THE ITEMS RETRIEVED FROM ANY NUMERIC DOMAIN. YOU WILL BE ASKED TO
INDICATE IF YOU WISH TO AVOID YOURSELF OF THIS FEATURE.

ADDRESS
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

DEPT-LOC
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

DEPT-MANAGER
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

DEPT-NAME
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

DEPT-NO
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ACCUMULATE A TOTAL OF RETRIEVED ITEMS IN THIS
DOMAIN? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

EMPLOYEE-NAME
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

EMPLOYEE-NO
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ACCUMULATE A TOTAL OF RETRIEVED ITEMS IN THIS
DOMAIN? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

JOIN-DAY
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.
DO YOU WISH TO ACCUMULATE A TOTAL OF RETRIEVED ITEMS IN THIS
DOMAIN? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

JOIN-MONTH
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

JOIN-YEAR
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

MARITAL-ST.
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.

SEX
DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TYPE Y FOR YES OR
N FOR NO. THE DEFAULT REPLY IS NO.
5. STAGE 5

STAGE 5 - EXTENSION OF DATABASE

ALTHOUGH YOUR DATABASE MAY ONLY BE EXTENDED ON A PERMANENT BASIS IN CONSULTATION WITH THE DATABASE ADMINISTRATOR, YOU MAY CREATE NEW DATA ITEMS WHICH ARE AVAILABLE ONLY DURING THIS SYSTEM SESSION. THE PROCESSING OF THESE TEMPORARY ITEMS WILL BE ENTIRELY UNDER YOUR CONTROL AND LATER YOU WILL BE GIVEN AN OPPORTUNITY TO SPECIFY YOUR PROCESSING REQUIREMENTS.

IT IS AT THIS STAGE YOU SHOULD CREATE ANY ITEMS REQUIRED FOR HOLDING INTERMEDIATE RESULTS OF CALCULATIONS OR FOR REMAINDERS ARISING FROM ARITHMETIC DIVISIONS.

DO YOU WISH TO CREATE ANY TEMPORARY DATA ITEMS?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO. :

DO YOU WISH TO CREATE ANY TEMPORARY ITEMS WHICH CONTAIN ALPHANUMERIC DATA?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO CREATE ANY TEMPORARY ITEMS WHICH CONTAIN NUMERIC DATA?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE TYPE THE NAME OF THE NUMERIC ITEM. :

OF HOW MANY DIGITS DOES WORK-ITEM CONSIST?
The default reply is 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER THAN AFTER THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR WORK-ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO. BY DEFAULT THE ITEM WILL INITIALLY CONTAIN ZERO.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE TYPE THE NAME OF THE NUMERIC ITEM. :

OF HOW MANY DIGITS DOES L-O-S-YEARS CONSIST?
The default reply is 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER THAN AFTER THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR L-O-S-YEARS?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO. BY DEFAULT THE ITEM WILL INITIALLY CONTAIN ZERO.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
PLEASE TYPE THE NAME OF THE NUMERIC ITEM.
L-O-S-MONTHS
OF HOW MANY DIGITS DOES L-O-S-MONTHS CONSIST?
THE DEFAULT REPLY IS 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER
THAN AFTER THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR L-O-S-MONTHS?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
BY DEFAULT THE ITEM WILL INITIALLY CONTAIN ZERO.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE TYPE THE NAME OF THE NUMERIC ITEM.
NO-EMPLOYEES
OF HOW MANY DIGITS DOES NO-EMPLOYEES CONSIST?
THE DEFAULT REPLY IS 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER
THAN AFTER THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR NO-EMPLOYEES?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER THE INITIAL VALUE FOR NO-EMPLOYEES.
THE VALUE MUST CONSIST OF DIGITS AND MAY INCLUDE A DECIMAL POINT, WHICH
MAY NOT BE THE LAST CHARACTER. THE SIGN, IF PRESENT, MUST BE THE FIRST
CHARACTER. BY DEFAULT THE SIGN IS POSITIVE.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE TYPE THE NAME OF THE NUMERIC ITEM.
AV-SER-YEARS
OF HOW MANY DIGITS DOES AV-SER-YEARS CONSIST?
THE DEFAULT REPLY IS 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER
THAN AFTER THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR AV-SER-YEARS?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
BY DEFAULT THE ITEM WILL INITIALLY CONTAIN ZERO.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
PLEASE TYPE THE NAME OF THE NUMERIC ITEM.
AV-SER-MONTHS
OF HOW MANY DIGITS DOES AV-SER-MONTHS CONSIST?
The default reply is 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER THAN AFTER THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR AV-SER-MONTHS?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
By default the item will contain zero.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE TYPE THE NAME OF THE NUMERIC ITEM.
MONTHS-SERVICE
OF HOW MANY DIGITS DOES MONTHS-SERVICE CONSIST?
The default reply is 18.

DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHER THAN THE RIGHTMOST DIGIT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

DO YOU WISH TO ENTER AN INITIAL VALUE FOR MONTHS-SERVICE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
By default the item will initially contain zero.

DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL FOR THIS ITEM?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HAVE YOU ANY MORE TEMPORARY NUMERIC ITEMS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

YOUR DATABASE MAY ALSO BE TEMPORARILY EXTENDED BY THE INCLUSION OF A COMPOUND DOMAIN FORMED BY CONCATENATING THE ITEMS FROM TWO OR MORE OTHER DOMAINS WITHIN THE DATABASE.
E.G. THE TEMPORARY DOMAIN ADDRESS MAY BE COMPOUNDED FROM THE EXISTING DOMAINS STREET, TOWN AND POSTCODE. COMPOUND DOMAINS ARE ALWAYS OF ALPHANUMERIC TYPE.

DO YOU WISH TO TEMPORARILY EXTEND YOUR DATABASE BY THE INCLUSION OF A COMPOUND DOMAIN?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE TYPE THE NAME OF THE COMPOUND DOMAIN WHICH YOU WISH TO CREATE.

PLEASE TYPE THE NAME OF THE FIRST DOMAIN TO BE CONCATENATED

PLEASE TYPE THE NAME OF THE NEXT DOMAIN TO BE CONCATENATED

HAVE YOU ANY MORE DOMAINS TO INCLUDE IN THIS CONCATENATION?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
PLEASE TYPE THE NAME OF THE NEXT DOMAIN TO BE CONCATENATED
JOIN-YEAR
HAVE YOU ANY MORE DOMAINS TO INCLUDE IN THIS CONCATENATION?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
:-N
COMPOUND DOMAIN JOIN-DATE CONTAINS 6 CHARACTERS AND IS
ALPHANUMERIC IN TYPE.
HAVE YOU ANY MORE DOMAINS TO CREATE?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
:-N
6. STAGE 6

STAGE 6 - DATABASE INCONSISTENCIES
YOUR PROBLEM MAKES USE OF MORE THAN ONE RELATION. IF, WHEN THE RELATIONS
ARE JOINED, A DATA INCONSISTENCY IN A DOMAIN USED AS A SEQUENCE KEY IS
DETECTED BY THE SYSTEM SUCH THAT THERE IS NOT A MATCHING KEY ITEM FOR
ALL RELATIONS, WHICH ONE OF THE FOLLOWING ACTIONS DO YOU WISH THE SYSTEM
TO TAKE?
1. IGNORE THE UNMATCHED ITEMS AND CONTINUE WITH THE NEXT DATA RETRIEVAL
2. CREATE DUMMY MATCHING KEY ITEMS AND CONTINUE PROCESSING ON THE
   ASSUMPTION THAT ALL NUMERIC NON-KEY ITEMS CONTAIN ZEROS AND ALL
   ALPHANUMERIC NON-KEY ITEMS CONTAIN BLANKS.
3. TERMINATE THE PROCESSING OF THE DATABASE.

PLEASE TYPE 1, 2 OR 3, THE DEFAULT REPLY IS 1.

DO YOU WANT THE SYSTEM TO PRINT A MESSAGE WHEN A MIS-MATCH CONDITION IS
DETECTED? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
7. STAGE 7

DO YOU WISH TO SELECT ONLY CERTAIN ITEMS FOR RETRIEVAL FROM THE DOMAINS IN YOUR RELATION(S)?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

STAGE 7 - SELECTION OF DATA FOR RETRIEVAL

YOU ARE ABOUT TO SPECIFY THE CONDITION(S) WHICH MUST BE SATISFIED BEFORE THE DATA IS SELECTED FOR RETRIEVAL FROM THE DATA BASE.

CONDITIONS MAY BE SIMPLE OR COMPOUND. A COMPOUND CONDITION IS MADE UP OF SEVERAL SIMPLE CONDITIONS LINKED BY THE CONJUNCTIONS AND/ OR. THE SYSTEM WILL PROMPT YOU TO BUILD UP COMPOUND CONDITIONS IN TERMS OF SIMPLE CONDITIONS.

EACH SIMPLE CONDITION CONSISTS OF THREE PARTS:

TWO OPERANDS SEPARATED BY AN OPERATOR.

THE FIRST OPERAND IS THE NAME OF THE DOMAIN TO WHICH THE ITEMS TO BE TESTED BELONG.

THE SECOND OPERAND SPECIFIES WITH WHAT THE FIRST OPERAND MUST BE COMPARED. IT MAY BE THE NAME OF A TEMPORARY ITEM, A LITERAL VALUE OR THE NAME OF THE DOMAIN TO WHICH THE ITEM BEING COMPARED BELONGS.

IF YOU ENTER AN ALPHANUMERIC LITERAL AS THE SECOND OPERAND YOU MUST ENCLOSE IT IN QUOTATION MARKS, E.G. "ALPHANUMERIC LITERAL".

THE OPERATOR SPECIFIES HOW THE COMPARISON BETWEEN THE FIRST AND SECOND OPERANDS IS TO BE MADE DURING THE CONDITION TEST.

THE OPERATOR MUST BE ONE OF THE FOLLOWING:

OPERATOR MEANING

.EQ. EQUAL TO
.LT. LESS THAN
.LE. LESS THAN OR EQUAL TO
.GT. GREATER THAN
.GE. GREATER THAN OR EQUAL TO
.NE. NOT EQUAL TO

THE FULL STOPS ARE AN INTEGRAL PART OF THE OPERATOR.

YOU MAY NEGATE ANY OF THE ABOVE OPERATORS BY INCLUDING THE WORD NOT IMMEDIATELY AFTER THE FIRST FULL STOP.

E.G.

.NOT GT. NOT GREATER THAN

HERE ARE SOME EXAMPLES OF SIMPLE CONDITIONS:

ACCOUNT-NUMBER .LT. TRANSACTION-NO
QTY-ON-HAND .NOT GT. 400
PART-NAME .EQ. "NAILS"

DATA RETRIEVAL

DO YOU WISH TO SPECIFY A COMPOUND CONDITION FOR THE ABOVE TASK?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

:N

PLEASE TYPE YOUR SIMPLE CONDITION.

.L-O-S-YEARS .GE. 10
8. STAGE 8

STAGE 8 - REPORT LAYOUT INTRODUCTION

PLEASE CHOOSE YOUR PAGE WIDTH. A NARROW PAGE OF 70 CHARACTERS IS
SUITABLE FOR TELETYPewriter OR VDU OUTPUT. A WIDE PAGE OF 120 CHARACTERS
IS FOR LINE PRINTER OUTPUT. PLEASE TYPE N FOR NARROW OR W FOR WIDE.
THE DEFAULT REPLY IS NARROW.

*:N

PLEASE TYPE THE MAXIMUM NUMBER OF LINES TO BE PRINTED ON A REPORT PAGE
THIS MUST BE A VALUE IN THE RANGE 40 TO 99. THE DEFAULT VALUE IS 60

*:45

WOULD YOU LIKE TO DESIGN THE FORMAT OF THE LINES OF YOUR REPORT OR WOULD
YOU PREFER ONLY TO SPECIFY WHICH ITEMS SHOULD BE INCLUDED IN THE OUTPUT
AND LEAVE THE SYSTEM TO ARRANGE THE EXACT FORMAT OF THE LINES?

IF YOU DESIGN YOUR OWN LAYOUT, YOU MAY INCLUDE TEXT INFORMATION AS WELL
AS DATA ITEMS INCLUDING THE DATE, TIME AND PAGE NUMBER. YOU MAY ALSO
CONTROL THE SPACING OF LINES AND INSERT EDITING CHARACTERS INTO DATA
ITEMS TO IMPROVE THEIR PRESENTATION.

THE SYSTEM WILL ARRANGE THE ITEMS FOR OUTPUT IN THE REQUIRED ORDER
ACROSS THE LINE WITH TWO BLANK SPACES BETWEEN ITEMS. A NEW LINE WILL BE
TAKEN WHEN THERE IS INSUFFICIENT ROOM AT THE END OF A LINE FOR THE
COMPLETE ITEM. ALL OUTPUT ITEMS WILL BE UNEDITED.

DO YOU WISH TO SPECIFY THE FORMAT OF YOUR REPORT? PLEASE TYPE Y FOR YES
OR N FOR NO. THE DEFAULT REPLY IS NO.

*:Y

YOU WILL SHORTLY BE ASKED TO DESIGN THE LINES WHICH COMprise THE TITLES,
HEADINGS AND DETAIL LINES OF YOUR REPORT. TO ASSIST WITH THIS THE NAMES
OF YOUR DATA DOMAINS AND TEMPORARY ITEMS WILL BE DISPLAYED TO YOU ONE AT
A TIME. YOU ARE THEN REQUIRED TO SELECT A ONE CHARACTER LABEL FROM THE
LIST SHOWN BELOW FOR USE LATER WHEN INDICATING THE PRINT POSITIONS OF
THE ITEM IN QUESTION. THE FOLLOWING CHARACTERS ARE VALID LABELS.
A E F G H I J K L N O Q T U V W X Y Z

YOU WILL HAVE NOTICED THAT NOT ALL ALPHABETIC CHARACTERS ARE AVAILABLE
FOR USE AS DATA LABELS. THIS IS BECAUSE SOME OF THEM HAVE SPECIAL
MEANINGS ASSOCIATED WITH EDITING DATA ITEMS PRIOR TO PRINTING. THE
EDITING FACILITIES WILL BE DESCRIBED TO YOU SHORTLY.

WHEN A DATA NAME IS DISPLAYED PLEASE TYPE THE CHARACTER YOU HAVE DECIDED
TO USE AS ITS LABEL. THE DEFAULT REPLY, MADE BY PRESSING THE ACCEPT KEY
ONLY, INDICATES THAT THE DATA DOES NOT APPEAR IN THE OUTPUT REPORT.

AV-SER-MONTHS
I=U

AV-SER-YEARS
*:U

DEPT-MANAGER
*:F

DEPT-NO
*:A

EMPLOYEE-NAME
*:E
When specifying the format of a line of the report, the print positions to be occupied by an item are indicated by typing the label character in the desired positions of the line. Blank characters before and between items should be indicated by typing the # character, but blank characters at the righthand end of a line may be omitted.

E.g. If K is the label assigned to part-code-domain items and U is the label assigned to unit-cost domain items and it is desired to print these two items in print positions 10-16 and 20-25 respectively, then the required format is indicated by the following line:

############KKKKKK###!!!!!!!

If more print positions are allocated to an alphanumeric item than the item actually requires, the item is left justified in the specified positions and padded out with blanks. If too few print positions are allocated to a data item then characters at the righthand end will be truncated.

Unless the editing feature to be described shortly is in use, the decimal point in the output format of a numeric item is assumed to be after the rightmost digit. Leading zeros are replaced by blanks up to but not including a zero before the implied decimal point. The decimal point location in the item to be output is aligned with that implied in the output format. If more print positions are allocated than the item requires, the unused positions before the decimal point will be filled with blanks. If too few print positions are allocated then the high order digits are truncated to make it fit.
STAGE 9 - EDITING

DATA ITEMS MAY BE EDITED PRIOR TO OUTPUT IN THE REPORT IN ORDER TO IMPROVE THE PRESENTATION. THE EDITING OPTIONS AVAILABLE VARY DEPENDING ON WHETHER THE ITEM TO BE EDITED IS ALPHANUMERIC OR NUMERIC.

TWO CHARACTERS ARE AVAILABLE FOR USE WITH ALPHANUMERIC ITEMS AND THEY ARE THE INSERTION CHARACTERS B AND 0.

INSERTION CHARACTERS MAY BE INSERTED ANYWHERE IN THE STRING OF LABEL CHARACTERS DEFINING THE PRINT POSITIONS OF AN ALPHANUMERIC DATA ITEM. THE 0 CHARACTER WILL PRINT AS A ZERO IN THE OUTPUT LINE, BUT A B CHARACTER WILL BE REPLACED BY A BLANK SPACE.

E.G. IF IT IS DESIRED THAT A 7 CHARACTER PART-CODE ITEM, WHICH HAS BEEN ASSIGNED THE PRINT LABEL K, SHOULD OCCUPY PRINT POSITIONS 6-14 AND HAVE A BLANK SPACE BETWEEN THE FIRST AND SECOND CHARACTERS AND A ZERO BETWEEN THE FIFTH AND SIXTH CHARACTERS, THEN THE PRINT-LINE FORMAT SHOULD BE TYPED AS FOLLOWS:

#####KBKKK0KK

IF THE PART-CODE ITEM CONTAINS THE VALUE "CNUT4DE" THEN THE EDITED VALUE WILL PRINT AS FOLLOWS:

C NUT40DE

WOULD YOU LIKE TO EXPERIMENT WITH THE EFFECT OF USING INSERTION CHARACTERS TO EDIT ALPHANUMERIC DATA ITEMS READY FOR OUTPUT? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

THE FOLLOWING MAY BE USED TO EDIT NUMERIC ITEMS PRIOR TO OUTPUT:

$ \* + - , CR DB B 0

THE FUNCTION OF EACH OF THE ABOVE EDITING SYMBOLS AND THE COMBINATIONS PERMITTED ARE SET OUT BELOW.

$ AND \ ARE CURRENCY SYMBOLS. A CURRENCY SYMBOL, IF USED, MUST APPEAR AT THE BEGINNING OF AN OUTPUT FORMAT SPECIFICATION. IT WILL CAUSE THE SELECTED CURRENCY SYMBOL TO PRINT IN FRONT OF THE MOST SIGNIFICANT DIGIT OF THE ITEM BEING EDITED. IT MAY ONLY BE PRECEDED BY A + OR A - CHARACTER.

* IS THE CHEQUE PROTECTION SYMBOL. IF USED, THIS CHARACTER MUST APPEAR AT THE BEGINNING OF AN OUTPUT FORMAT SPECIFICATION. IT WILL CAUSE UNUSED HIGH ORDER PRINT POSITIONS OF THE ITEM TO BE FILLED WITH *S. ONLY THE + OR - CHARACTER MAY PRECEDE IT.

CR AND DB ARE REPORT SIGNS. EITHER CR OR DB MAY APPEAR IN THE TWO RIGHT-MOST CHARACTERS OF THE OUTPUT FORMAT FOR A NUMERIC ITEM. IF THE ITEM BEING EDITED IS NEGATIVE THEN THE SPECIFIED REPORT SIGN CHARACTERS WILL PRINT IN THE DESIRED POSITIONS. IF THE ITEM BEING EDITED IS POSITIVE THEN THE TWO PRINT POSITIONS OCCUPIED BY THE REPORT SIGN WILL BE SET TO BLANKS.

0, AND . ARE INSERTION CHARACTERS. THESE CHARACTERS WILL PRINT IN THE DESIGNATED POSITIONS OF THE EDITED ITEM WITH THE EXCEPTION OF THE 0 WHICH WILL BE REPLACED BY A BLANK. THE FULL STOP, IF PRESENT, MAY ONLY OCCUR ONCE AS IT REPRESENTS THE DECIMAL POINT LOCATION. THE FULL STOP MAY NOT APPEAR IN THE RIGHT-MOST PRINT POSITION OF AN EDITED ITEM AND MUST BE PRECEDED BY AT LEAST ONE LABEL CHARACTER.

HERE ARE SOME EXAMPLES OF EDITED NUMERIC ITEMS:

<table>
<thead>
<tr>
<th>EDITED FORMAT TO BE EDITED</th>
<th>VALUE OF ITEM</th>
<th>EDITED RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-SAAA,000</td>
<td>-23</td>
<td>-23,000</td>
</tr>
<tr>
<td>*EEEEECR</td>
<td>4795</td>
<td>**4795</td>
</tr>
<tr>
<td>FF,FF,FF</td>
<td>1347.63</td>
<td>1,347.63</td>
</tr>
<tr>
<td>GBBGOGGG</td>
<td>27641</td>
<td>27641</td>
</tr>
</tbody>
</table>

WOULD YOU LIKE TO EXPERIMENT WITH THE EFFECT OF EDITING NUMERIC DATA ITEMS READY FOR OUTPUT?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
STAGE 10 - DETAIL LINE(S) SPECIFICATION

FOR EACH RETRIEVAL FROM THE DATABASE ONE OR MORE DETAIL LINES MAY BE OUTPUT. A DETAIL LINE MAY CONTAIN TEXT AND/OR INCLUDE ANY ITEM FROM THE CURRENTLY RETRIEVED SET OF DATA AND/OR ANY TEMPORARY ITEM. IF MORE THAN ONE DETAIL LINE IS TO BE OUTPUT FOR EACH RETRIEVAL, BLANK LINES MAY BE INCLUDED BETWEEN THEM BY PRESSING ONLY THE ACCEPT KEY WHEN PROMPTED TO ENTER THE NEXT FORMAT LINE. THE FIRST LINE MAY NOT BE BLANK.

ANY LINE OF YOUR REPORT MAY CONTAIN TEXT INFORMATION. IN ORDER TO DISTINGUISH TEXT CHARACTERS FROM LABEL CHARACTERS DENOTING THE PRINT POSITIONS OF DATA ITEMS, THE TEXT IS ENCLOSED IN QUOTATION MARKS. THE QUOTATION MARKS ARE REPLACED BY BLANK CHARACTERS IN THE PRINTED REPORT.

E.G. A DATA ITEM UNIT-PRICE IS ASSIGNED THE PRINT LABEL 'U' AND IS TO BE PRINTED IN PRINT POSITIONS 15-20 TOGETHER WITH THE NAME OF THE ITEM IN PRINT POSITIONS 2-11. THE OUTPUT FORMAT FOR THE ABOVE LINE SHOULD BE TYPED AS FOLLOWS:

"UNIT PRICE"##SUU.UU

HOW MANY BLANK LINES WOULD YOU LIKE TO APPEAR BETWEEN THE PREVIOUS LINE OF OUTPUT AND THE FIRST DETAIL LINE OF A RETRIEVAL? PLEASE TYPE A NUMBER IN THE RANGE 0 TO 3. THE DEFAULT REPLY IS 0.

PLEASE ENTER THE FORMAT FOR THE FIRST DETAIL LINE.

HAVE YOU ANY MORE DETAIL LINES TO ENTER?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

IF THE DETAIL LINES FROM ONE RETRIEVAL DUPLICATE THOSE FROM AN EARLIER RETRIEVAL DO YOU WISH THE DUPLICATES TO BE IGNORED?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
STAGE 11 - EXTRA DATA ITEMS - DATE, TIME, PAGE NO.
YOU HAVE ALREADY INDICATED WHICH DATA ITEMS YOU WISH TO OUTPUT IN YOUR DETAIL LINE(S), IN ADDITION THERE ARE THREE OTHERS THAT YOU MAY INCLUDE IF YOU OPT TO HAVE A REPORT TITLE OR PAGE HEADING. THESE ARE THE DATE AND TIME AT WHICH YOUR REPORT PROGRAM BEGINS TO EXECUTE AND THE REPORT PAGE NUMBER.
DATE AND TIME EACH OCCUPY 8 PRINT POSITIONS WHILE PAGE NUMBER OCCUPIES 4. THE FOLLOWING STRINGS OF CHARACTERS MUST BE USED TO DENOTE THE PRINT POSITIONS OF THESE SPECIAL ITEMS:

<table>
<thead>
<tr>
<th>ITEM</th>
<th>PRINT STRING</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DD/MM/YY</td>
<td>DD=DAY MM=MONTH YY=YEAR</td>
</tr>
<tr>
<td>TIME</td>
<td>HH/MM/SS</td>
<td>HH=HOURS MM=MINUTES SS=SECONDS</td>
</tr>
<tr>
<td>PAGE NO.</td>
<td>PPPP</td>
<td></td>
</tr>
</tbody>
</table>
STAGE 12 - REPORT TITLE SPECIFICATION

DO YOU WISH TO HAVE A TITLE PAGE ON YOUR REPORT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

ON WHICH LINE OF THE PAGE WOULD YOU LIKE THE TITLE TO START? PLEASE TYPE A NUMBER IN THE RANGE 1 TO 22. THE DEFAULT REPLY IS 1.

THE TITLE MAY CONSIST OF SEVERAL LINES AND INCLUDE THE DATE, TIME AND PAGE NUMBER ITEMS AS WELL AS TEXT. A BLANK LINE MAY BE INCLUDED IN THE TITLE BY PRESSING ONLY THE ACCEPT KEY WHEN PROMPTED TO ENTER ANOTHER FORMAT LINE. THE FIRST LINE MAY NOT BE BLANK.

PLEASE ENTER THE FORMAT OF THE FIRST LINE OF THE TITLE.

HAVE YOU ANY MORE TITLE LINES TO ENTER?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER THE FORMAT OF THE NEXT LINE OF THE TITLE.

HAVE YOU ANY MORE TITLE LINES TO ENTER?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER THE FORMAT OF THE NEXT LINE OF THE TITLE.

HAVE YOU ANY MORE TITLE LINES TO ENTER?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER THE FORMAT OF THE NEXT LINE OF THE TITLE.

HAVE YOU ANY MORE TITLE LINES TO ENTER?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER THE FORMAT OF THE NEXT LINE OF THE TITLE.

HAVE YOU ANY MORE TITLE LINES TO ENTER?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
STAGE 13 - PAGE HEADING SPECIFICATION

Each page of your report may have a page heading. The text at the top of each report page will be the same, although if you choose to include data items their values may vary from page to page. As the data to be retrieved is available in an ordered sequence based on the items in the 2 key domains, you may in addition to the page heading specify sequence break headings.

A different sequence break heading may be specified for each key domain except the minor key, which changes with each retrieval. The sequence break heading will print after the page heading at the top of each page and again lower down when a sequence break in an item retrieved from a non-minor key domain is detected.

At the top of the page all the sequence break headings are output in order from major to penultimate minor key sequence. Lower down the page a sequence break at one level will cause that sequence break heading to be printed and also those at lower levels, if any.

Do you wish to specify a page heading for each page of your report? Please type Y for yes or N for no. The default reply is NO.

A page heading consists of one or more lines of text and/or may include any item from a currently retrieved set of data, any temporary item and/or the date, time or page number items. A blank line may be included in the page heading by pressing only the accept key when prompted to enter another format line.

On which line of the page would you like the page heading to start? Please type a number in the range 1 to 10. The default reply is 1.

The first line may not be blank.

Please enter the format of the first line of the page heading.

Have you any more page heading lines to enter? Please type Y for yes or N for no. The default reply is NO.

Please enter the format of the next line of page heading.

Have you any more page heading lines to enter? Please type Y for yes or N for no. The default reply is NO.
STAGE 14 - SEQUENCE BREAK HEADING SPECIFICATION

Do you wish to specify sequence break headings in your report? Please type Y for YES or N for NO. The default reply is NO.

A sequence break heading consists of one or more lines of text and/or may include any item from a currently retrieved set of data or any temporary item. A blank line may be included in the sequence break heading by pressing only the accept key when prompted to enter another format line.

Sequence break heading for key domain Dept-No

How many blank lines would you like to appear between the previous line of output and the first line of this heading? Please type a number in the range 0 to 3. The default reply is 0.

The first line may not be blank.

Please enter the format of the first line of this heading.

Have you any more lines to enter for this heading? Please type Y for YES or N for NO. The default reply is NO.

Please enter the format of the next line of this heading.

Have you any more lines to enter for this heading? Please type Y for YES or N for NO. The default reply is NO.

Please enter the format of the next line of this heading.

Have you any more lines to enter for this heading? Please type Y for YES or N for NO. The default reply is NO.

Please enter the format of the next line of this heading.

Have you any more lines to enter for this heading? Please type Y for YES or N for NO. The default reply is NO.

Have you any more lines to enter for this heading? Please type Y for YES or N for NO. The default reply is NO.

Have you any more lines to enter for this heading? Please type Y for YES or N for NO. The default reply is NO.
STAGE 15 - SUB-TOTAL LINE(S) SPECIFICATION

You have asked for the totals of certain numeric items to be accumulated during the processing of your data. Shortly you will be asked to specify the format in which these totals are to be output at the end of your report.

Meanwhile, as the data is retrieved in an ordered sequence, you may ask for sub-totals to be output when a sequence change is detected in the data retrieved from any key domain except the minor key, which changes with each retrieval.

Sub-totals are output in order from penultimate minor to major key sequence. A sequence change at an intermediate level will cause the sub-totals for that key and for all the keys minor to it to be output as well.

Do you wish to specify sequence break sub-totals in your report? Please type Y for yes or N for no. The default reply is NO.

A sequence break sub-total consists of one or more lines of text and/or may include any non-accumulated item, either temporary or from the previous set of retrieved data, as well as the sub-totals of the items which have been accumulated. A blank line may be included in a sequence break sub-total by pressing only the accept key when prompted to enter another format line.

Sub-total output for a sequence break in key-domain DEPT-NO how many blank lines would you like to appear between the previous line of output and the first line of the sub-total? Please type a number in the range 0 to 3. The default reply is 0.

The first line may not be blank.

Please enter the format of the first line of this sub-total output.

Have you any more lines to enter for this sub-total output? Please type Y for yes or N for no. The default reply is NO.

Please enter the format of the next line of this sub-total output.

Have you any more lines to enter for this sub-total output? Please type Y for yes or N for no. The default reply is NO.
STAGE 16 - TOTAL LINE(S) SPECIFICATION

THE ACCUMULATED TOTALS OUTPUT CONSISTS OF ONE OR MORE LINES OF TEXT AND/OR MAY INCLUDE ANY NON-ACCUMULATED ITEM FROM THE PREVIOUS SET OF RETRIEVED DATA OR ANY TEMPORARY ITEM AS WELL AS THE TOTALS OF THE ITEMS WHICH HAVE BEEN ACCUMULATED. A BLANK LINE MAY BE INCLUDED IN THE TOTALS BY PRESSING ONLY THE ACCEPT KEY WHEN PROMPTED TO ENTER ANOTHER FORMAT LINE.

DO YOU WISH TO SPECIFY TOTALS OUTPUT IN YOUR REPORT? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

HOW MANY BLANK LINES WOULD YOU LIKE TO APPEAR BETWEEN THE PREVIOUS LINE OF OUTPUT AND THE FIRST LINE OF TOTALS? PLEASE TYPE A NUMBER IN THE RANGE 0 TO 3. THE DEFAULT REPLY IS 0.

THE FIRST LINE MAY NOT BE BLANK PLEASE ENTER THE FORMAT OF THE FIRST TOTAL LINE. :

"COMPANY TOTAL "TTTT"EMPLOYEES"

HAVE YOU ANY MORE TOTAL LINES TO ENTER? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

THE FORMAT OF THE NEXT TOTAL LINE.

"AVERAGE LENGTH OF SERVICE "UU"YEARS"UU"MONTHS"

HAVE YOU ANY MORE TOTAL LINES TO ENTER? PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
STAGE 17 - VIEW OF SAMPLE REPORT PAGE

You are about to be offered the opportunity of seeing a sample of a page of your report. The sample page will show you the arrangement of the various lines whose format you have just specified, but will not provide actual values for data items.

Would you like to see a sample of an output page? Please type Y for yes or N for no. The default reply is NO.

Y
X.Y.Z. MANUFACTURING COMPANY LIMITED
LONG SERVICE REPORT
CONFIDENTIAL
LONG SERVICE REPORT FOR THE YEAR ENDING 31ST DECEMBER 1977

DEPARTMENT NO. AA  DEPARTMENT MANAGER  FFFFFFFFFFFF

<table>
<thead>
<tr>
<th>EMPLOYEE NO.</th>
<th>EMPLOYEE NAME</th>
<th>DATE OF JOINING</th>
<th>LENGTH OF SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
<tr>
<td>NNNN</td>
<td>EEEEEEEEEEEEEEEE</td>
<td>JJ  JJ  JJ</td>
<td>YY  XX</td>
</tr>
</tbody>
</table>

DEPARTMENT NO. AA  NO. OF EMPLOYEES TTTT  AVERAGE SERVICE VV  III

COMPANY TOTAL  TTTT EMPLOYEES
AVERAGE LENGTH OF SERVICE  VV YEARS III MONTHS

ARE YOU SATISFIED WITH THE PAGE LAYOUT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
:: Y
STAGE 18 - OWN CODE PROCESSING

You are about to be given the opportunity of supplementing the standard system processing of your data by specifying your own calculations and/or data movements. These should include the processing of temporary items.

The option to specify your own supplementary processing will be available at the following points during the execution of the COBOL program:

- At the start of processing before any data is retrieved,
- Immediately after a data retrieval,
- At a sequence break on a specific non-minor key domain item,
- Prior to printing any group of heading, detail, sub-total or total lines.

Do you wish to take advantage of the above facilities? Please type Y for yes or N for no. The default reply is NO. ☐ Y

Four arithmetic statements are available for use in your own calculations, one for each of the arithmetic operations: addition, subtraction, multiplication and division. Each statement requires three operands and takes the following form:

- ADD OPERAND-1, OPERAND-2 GIVING OPERAND-3
- SUBTRACT OPERAND-1 FROM OPERAND-2 GIVING OPERAND-3
- MULTIPLY OPERAND-1 BY OPERAND-2 GIVING OPERAND-3
- DIVIDE OPERAND-1 BY OPERAND-2 GIVING OPERAND-3

The division remainder may, if desired, be saved in a named domain or temporary item. OPERAND-3 may be the name of any temporary item or the name of a domain to which the item being calculated belongs. OPERAND-1 and OPERAND-2 may be either a numeric literal, the name of a temporary item or the name of a domain to which the item being calculated belongs.

E.g.
- MULTIPLY QUANTITY BY PRICE GIVING COST
- DIVIDE PENCE BY 100 GIVING POUNDS

One data movement statement is available for your own data movements. This statement requires two operands and has the following form:

- MOVE OPERAND-1 TO OPERAND-2

OPERAND-2 is the receiving operand and may be the name of a temporary item or the name of a domain to which the receiving item belongs. OPERAND-1 is the source item and may be a literal, a temporary item or the name of a domain to which the item being moved belongs. OPERAND-1 and OPERAND-2 must be of the same type, either both numeric or both alphanumeric.

E.g.
- MOVE "NOT MARRIED" TO SPOUSE-NAME
- MOVE QTY-ON-ORDER TO QTY-ON-HAND
- MOVE 1.65 TO UNIT-PRICE

When alphanumeric data is moved the characters from the source item are transferred to the corresponding positions of the receiving item, starting with the left-most. If the receiving item is the shorter of the two items then excess characters from the source item are ignored. If the receiving item is the longer the excess characters are filled with blanks.
WHEN NUMERIC DATA IS MOVED THE DECIMAL POINTS IN THE SOURCE ITEM AND RECEIVING ITEM ARE ALIGNED. IF THERE ARE FEWER DIGITS BEFORE OR AFTER THE DECIMAL POINT IN THE RECEIVING ITEM THEN THE EXCESS DIGITS ARE TRUNCATED. IF THERE ARE MORE DIGITS BEFORE OR AFTER THE DECIMAL POINT IN THE RECEIVING ITEM THEN THE EXCESS POSITIONS ARE SET TO CONTAIN ZERO.

AS YOU HAVE TAKEN ADVANTAGE OF THE SYSTEMS ACCUMULATED TOTALS OPTION YOU MAY, DURING YOUR OWN PROCESSING, HAVE ACCESS TO THE TOTALS APPLICABLE AT A SPECIFIC KEY SEQUENCE BREAK OR AFTER ALL THE DATA HAS BEEN RETRIEVED.

IN ORDER TO DISTINGUISH THE TOTALS OF ITEMS FROM A GIVEN DOMAIN FROM THE PREVIOUSLY RETRIEVED ITEM FROM THE SAME DOMAIN, THE DOMAIN NAME IS IMMEDIATELY PRECEDED BY AN * CHARACTER WHEN THE TOTAL IS REQUIRED.

E.G. A RELATION CONTAINS ITEMS FROM NUMERIC DOMAINS CALLED STOCK-VALUE AND QUANTITY AND IS SEQUENCED ON ITEMS IN KEY DOMAINS CALLED PART-CLASS AND PART-NO. IF THE ACCUMULATED TOTALS OPTION HAS BEEN REQUESTED FOR THE STOCK-VALUE AND QUANTITY ITEMS, THEN A TEMPORARY ITEM CALLED AVERAGE-PRICE MAY BE CALCULATED EACH TIME A SEQUENCE CHANGE OCCURS IN THE PART-CLASS KEY ITEM OF THE RETRIEVED DATA BY THE FOLLOWING OWN CODE STATEMENT:

```
DIVIDE *STOCK-VALUE BY *QUANTITY GIVING AVERAGE-PRICE
```

N.B. THIS SPECIAL TOTAL FORM OF A DATA NAME MAY NOT BE USED AS OPERAND-3 IN THE ARITHMETIC STATEMENTS NOR AS OPERAND-2 IN A MOVE STATEMENT.

BEFORE ENTERING ANY OWN CODE STATEMENTS YOU WILL BE GIVEN THE OPPORTUNITY TO SPECIFY WHAT CONDITIONS, IF ANY, MUST BE SATISFIED BEFORE THE STATEMENT IS OBEYED.

N.B. THE OWN CODE STATEMENTS AT EACH PROCESSING POINT ARE OBEYED IN THE ORDER IN WHICH YOU ENTER THEM.

WOULD YOU LIKE TO BE REMINDED OF HOW TO SPECIFY A CONDITION?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

AT THE START OF PROCESSING BEFORE ANY DATA IS RETRIEVED HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

IMMEDIATELY AFTER A DATA RETRIEVAL HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

IS THE OWN CODE STATEMENT THAT YOU ARE ABOUT TO ENTER CONDITIONAL?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER YOUR OWN CODE STATEMENT.

```
SUBTRACT JOIN-YEAR FROM 77 GIVING WORK-ITEM
```

HAVE YOU ANY MORE OWN CODE STATEMENTS TO ENTER AT THE ABOVE POINT?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

IS THE OWN CODE STATEMENT THAT YOU ARE ABOUT TO ENTER CONDITIONAL?

PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.

PLEASE ENTER YOUR OWN CODE STATEMENT.

```
MULTIPLY WORK-ITEM BY 12 GIVING MONTHS-SERVICE
```
HAVE YOU ANY MORE OWN CODE STATEMENTS TO ENTER AT THE ABOVE POINT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::Y
IS THE OWN CODE STATEMENT THAT YOU ARE ABOUT TO ENTER CONDITIONAL?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PLEASE ENTER YOUR OWN CODE STATEMENT.
::SUBTRACT JOIN-MONTH FROM 13 GIVING WORK-ITEM
HAVE YOU ANY MORE OWN CODE STATEMENTS TO ENTER AT THE ABOVE POINT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::Y
IS THE OWN CODE STATEMENT THAT YOU ARE ABOUT TO ENTER CONDITIONAL?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PLEASE ENTER YOUR OWN CODE STATEMENT.
::ADD WORK-ITEM, MONTHS-SERVICE GIVING MONTHS-SERVICE
HAVE YOU ANY MORE OWN CODE STATEMENTS TO ENTER AT THE ABOVE POINT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::Y
IS THE OWN CODE STATEMENT THAT YOU ARE ABOUT TO ENTER CONDITIONAL?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PLEASE ENTER YOUR OWN CODE STATEMENT.
::DIVIDE MONTHS-SERVICE BY 12 GIVING L-O-S-YEARS
IF YOU WISH TO SAVE THE REMAINDER PLEASE TYPE THE NAME OF THE DOMAIN OR TEMPORARY ITEM WHERE IT IS TO BE STORED. BY DEFAULT THE REMAINDER WILL NOT BE SAVED.
::L-O-S-MONTHS
HAVE YOU ANY MORE OWN CODE STATEMENTS TO ENTER AT THE ABOVE POINT?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
AT A SEQUENCE BREAK IN THE DEPT-NO KEY DOMAIN ITEM
HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PRIOR TO PRINTING A PAGE HEADING
HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PRIOR TO PRINTING THE DEPT-NO SEQUENCE BREAK HEADING
HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PRIOR TO PRINTING THE DETAIL LINE(S)
HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PRIOR TO PRINTING THE DEPT-NO SUBTOTALS
HAVE YOU ANY OWN CODE PROCESSING TO SPECIFY FOR THE ABOVE POINT DURING THE DATA PROCESSING RUN?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::Y
IS THE OWN CODE STATEMENT THAT YOU ARE ABOUT TO ENTER CONDITIONAL?
PLEASE TYPE Y FOR YES OR N FOR NO. THE DEFAULT REPLY IS NO.
::N
PLEASE ENTER YOUR OWN CODE STATEMENT.

: *DIVIDE *MONTHS-SERVICE BY *NO-EMPLOYEES GIVING WORK-ITEM
If you wish to save the remainder please type the name of the domain or temporary item where it is to be stored. By default the remainder will not be saved.

: *

Have you any more own code statements to enter at the above point? Please type Y for Yes or N for No. The default reply is No.

: -Y

Is the own code statement that you are about to enter conditional? Please type Y for Yes or N for No. The default reply is No.

: -N

Please enter your own code statement.

: *DIVIDE WORK-ITEM BY 12 GIVING AV-SER-YEARS
If you wish to save the remainder please type the name of the domain or temporary item where it is to be stored. By default the remainder will not be saved.

: *-

Have you any more own code statements to enter at the above point? Please type Y for Yes or N for No. The default reply is No.

: -N

Prior to printing the total line(s)
Have you any own code processing to specify for the above point during the data processing run? Please type Y for Yes or N for No. The default reply is No.

: -Y

Is the own code statement that you are about to enter conditional? Please type Y for Yes or N for No. The default reply is No.

: -N

Please enter your own code statement.

: -DIVIDE *MONTHS-SERVICE BY *NO-EMPLOYEES GIVING WORK-ITEM
If you wish to save the remainder please type the name of the domain or temporary item where it is to be stored. By default the remainder will not be saved.

: -

Have you any more own code statements to enter at the above point? Please type Y for Yes or N for No. The default reply is No.

: -Y

Is the own code statement that you are about to enter conditional? Please type Y for Yes or N for No. The default reply is No.

: -N

Please enter your own code statement.

: -DIVIDE WORK-ITEM BY 12 GIVING AV-SER-YEARS
If you wish to save the remainder please type the name of the domain or temporary item where it is to be stored. By default the remainder will not be saved.

: -AV-SER-MONTHS

Have you any more own code statements to enter at the above point? Please type Y for Yes or N for No. The default reply is No.

: -N

You have now fully specified your data and your problem and the system is ready to generate the COBOL program which will provide you with your answers.
IF YOU HAVE ANY COMMENTS ABOUT ANY ASPECT OF THIS SYSTEM PLEASE ENTER THEM LINE BY LINE AS YOU RECEIVE THE INVITATION TO TYPE. A BLANK RESPONSE WILL BE TAKEN TO INDICATE THE END OF YOUR COMMENTS.

END OF SESSION, GOODBYE. PLEASE TYPE %FINISH TO EXIT FROM THE SYSTEM.

%FINISH
19. CREATE CATALOGUE

CATALOGUE CREATION MACRO.
THIS MACRO CREATES AN EMPTY CATALOGUE AND ESTABLISHES THE SECURITY
SYSTEM FOR PROTECTING ACCESS TO THE FILE DESCRIPTIONS WHICH THE
CATALOGUE WILL EVENTUALLY CONTAIN.
PLease type the password to be used by persons entitled to amend the
catalogue. The password may be up to 8 characters long and the default
value is blank. Characters in excess of 8 will be ignored.
\$=CATPSWD
Please type the number of security levels by which access to file data
is to be protected. The number must be greater than zero and less than
13. The default reply is 12.
\$=40
40 INVALID SECURITY LEVEL. PLEASE ENTER CORRECT VALUE.
\$=4
EMPTY CATALOGUE WITH 4 SECURITY LEVELS CREATED ON
21/12/78 AT 14/53/24 AND READY FOR USE.

Catalogue Control Record

$0,1,CATPSWD,4,21/12/78,14/53/24,
20. CHANGE CATALOGUE PASSWORD

MACRO TO CHANGE CATALOGUE PASSWORD.
PLEASE TYPE THE OLD PASSWORD.
; -CATPWORD
CATPWORD IS AN INVALID PASSWORD. TASK ABANDONED.

MACRO TO CHANGE CATALOGUE PASSWORD.
PLEASE TYPE THE OLD PASSWORD.
; -CATPWORD
PLEASE TYPE THE NEW PASSWORD OF UP TO 8 CHARACTERS.
; -CATAPASS
NEW PASSWORD IN OPERATION ON 27/12/78 AT 15/20/35.
CATALOGUE LIBRARIAN,
THIS MACRO UPDATES THE DATABASE CATALOGUE OF FILE DESCRIPTIONS.
PLEASE TYPE THE CATALOGUE PASSWORD
+$KATAPASS
$KATAPASS IS AN INVALID PASSWORD. RUN TERMINATED.

CATALOGUE LIBRARIAN,
THIS MACRO UPDATES THE DATABASE CATALOGUE OF FILE DESCRIPTIONS.
PLEASE ENTER THE CATALOGUE PASSWORD
+$CATAPASS
DO YOU WISH TO ADD A NEW FILE DESCRIPTION OR DELETE AN EXISTING ONE?
PLEASE TYPE ADD OR DELETE, AT END TYPE FINISH.

DELETE
DELETE IS AN INVALID RESPONSE
PLEASE TYPE ADD OR DELETE, AT END TYPE FINISH.

DELETE
PLEASE TYPE THE NAME OF THE FILE WHOSE DESCRIPTION YOU WISH TO DELETE.

STAFF
PLEASE TYPE PASSWORD FOR TOP SECURITY LEVEL FOR FILE
STAFF

STAFFPASS
STAFFPASS IS NOT AN ACCEPTABLE PASSWORD. REQUEST CANCELLED.
DO YOU WISH TO ADD A NEW FILE DESCRIPTION OR DELETE AN EXISTING ONE?
PLEASE TYPE ADD OR DELETE, AT END TYPE FINISH.

DELETE
PLEASE TYPE THE NAME OF THE FILE WHOSE DESCRIPTION YOU WISH TO DELETE.

STAFF
PLEASE TYPE PASSWORD FOR TOP SECURITY LEVEL FOR FILE
STAFF

STAFF!
CATALOGUE UPDATED ON 21/11/78 AT 19/51/19
DO YOU WISH TO ADD A NEW FILE DESCRIPTION OR DELETE AN EXISTING ONE?
PLEASE TYPE ADD OR DELETE, AT END TYPE FINISH.

FINISH
23. LIBPRELIM AND LIBADD

DO YOU WISH TO ADD A NEW FILE DESCRIPTION OR DELETE AN EXISTING ONE?
PLEASE TYPE ADD OR DELETE, AT END TYPE FINISH.

ADD
PLEASE TYPE THE NAME OF THE FILE WHOSE DESCRIPTION YOU
WISH TO ADD:
DEPARTMENT
PLEASE TYPE PASSWORD FOR TOP SECURITY LEVEL FOR FILE
DEPARTMENT
DEPT1
PLEASE TYPE THE NUMBER OF FIELDS IN A RECORD OF FILE
4
 PLEASE TYPE THE PASSWORD FOR SECURITY LEVEL 2
DEPT2
PLEASE TYPE THE PASSWORD FOR SECURITY LEVEL 3
DEPT3
PLEASE TYPE THE PASSWORD FOR SECURITY LEVEL 4
DEPT4
PLEASE TYPE UP TO 70 CHARACTERS OF DESCRIPTION ABOUT FILE
DEPARTMENT DESCRIPTION RELATION
ON WHICH STORAGE MEDIUM IS THIS FILE?
PAPER TAPE
PLEASE TYPE 2 CHARACTERS, MT FOR MAGNETIC TAPE
OR FOR PUNCHED CARDS, PT FOR PAPER TAPE
ED FOR EXCHANGEABLE DISC STORE
MT
PLEASE ENTER BLOCKSIZE
396
PLEASE TYPE MAXIMUM NO. OF CHARACTERS IN A RECORD
66
PLEASE TYPE THE FILE LABEL WHICH MAY BE UP TO 12
CHARACTERS LONG. EXCESS CHARACTERS WILL BE IGNORED.
DEPARTMENT
PLEASE TYPE NO. OF SEQUENCE KEY FIELDS
1
YOU ARE ABOUT TO BE ASKED FOR DETAILS OF FIELDS WITHIN A RECORD
WHERE POSSIBLE PLEASE ENTER FIELD NAMES IN ASCENDING START POSITION
AS THIS MAKES FOR MORE EFFICIENT PROCESSING
PLEASE TYPE THE NAME OF FIELD NO. 1
DEPT-NO
WHICH SECURITY LEVEL DO YOU WISH TO ASSIGN TO DEPT-NO?
PLEASE TYPE A NUMBER IN THE RANGE 1 TO 4. 1 = HIGHEST LEVEL
4 = LOWEST LEVEL
4
IN WHICH CHARACTER POSITION DOES THE FIELD START?
1 IS THE FIRST CHARACTER POSITION IN THE RECORD
5
HOW MANY CHARACTER POSITIONS DOES THE FIELD OCCUPY?
2
DOES THIS FIELD CONTAIN NUMERIC OR ALPHANUMERIC DATA?
PLEASE TYPE N FOR NUMERIC OR A FOR ALPHANUMERIC
N
PLEASE TYPE THE POSITION OF THE DECIMAL POINT
RELATIVE TO THE RIGHHAND CHARACTER OF THE FIELD
TYPE L FOR LEFT OR R FOR RIGHT FOLLOWED BY AN UNSIGNED INTEGER
L0
PLEASE TYPE FIELD KEY NO. IN THE RANGE 0 TO 9
0 DENOTES A NON-KEY FIELD, 1 DENOTES THE MAJOR KEY

PLEASE INDICATE SEQUENCE ORDER. TYPE A FOR ASCENDING OR D FOR DESCENDING

PLEASE TYPE UP TO 70 CHARACTERS OF DESCRIPTION ABOUT THE FIELD

PLEASE TYPE THE NAME OF FIELD NO. 1

WHICH SECURITY LEVEL DO YOU WISH TO ASSIGN TO DEPT-NAME?
PLEASE TYPE A NUMBER IN THE RANGE 1 TO 4. 1=HIGHEST LEVEL
4 = LOWEST LEVEL

IN WHICH CHARACTER POSITION DOES THE FIELD START?
1 IS THE FIRST CHARACTER POSITION IN THE RECORD

HOW MANY CHARACTER POSITIONS DOES THE FIELD OCCUPY?

DOES THIS FIELD CONTAIN NUMERIC OR ALPHANUMERIC DATA?
PLEASE TYPE N FOR NUMERIC OR A FOR ALPHANUMERIC

PLEASE TYPE FIELD KEY NO. IN THE RANGE 0 TO 9
0 DENOTES A NON-KEY FIELD, 1 DENOTES THE MAJOR KEY

CAN FIELD BE TREATED AS A DOMAIN? TYPE Y FOR YES OR N FOR NO

PLEASE TYPE UP TO 70 CHARACTERS OF DESCRIPTION ABOUT FIELD

PLEASE TYPE THE NAME OF FIELD NO. 2

WHICH SECURITY LEVEL DO YOU WISH TO ASSIGN TO DEPT-LOC?
PLEASE TYPE A NUMBER IN THE RANGE 1 TO 4. 1=HIGHEST LEVEL
4 = LOWEST LEVEL

IN WHICH CHARACTER POSITION DOES THE FIELD START?
1 IS THE FIRST CHARACTER POSITION IN THE RECORD

HOW MANY CHARACTER POSITIONS DOES THE FIELD OCCUPY?

DOES THIS FIELD CONTAIN NUMERIC OR ALPHANUMERIC DATA?
PLEASE TYPE N FOR NUMERIC OR A FOR ALPHANUMERIC

PLEASE TYPE FIELD KEY NO. IN THE RANGE 0 TO 9
0 DENOTES A NON-KEY FIELD, 1 DENOTES THE MAJOR KEY

CAN FIELD BE TREATED AS A DOMAIN? TYPE Y FOR YES OR N FOR NO

PLEASE TYPE UP TO 70 CHARACTERS OF DESCRIPTION ABOUT THE FIELD

PLEASE TYPE THE NAME OF FIELD NO. 3

WHICH SECURITY LEVEL DO YOU WISH TO ASSIGN TO DEPT-MANAGER?
PLEASE TYPE A NUMBER IN THE RANGE 1 TO 4. 1=HIGHEST LEVEL
4 = LOWEST LEVEL
IN WHICH CHARACTER POSITION DOES THE FIELD START?
1 IS THE FIRST CHARACTER POSITION IN THE RECORD
:**47
HOW MANY CHARACTER POSITIONS DOES THE FIELD OCCUPY?
:**20
DOES THIS FIELD CONTAIN NUMERIC OR ALPHANUMERIC DATA?
PLEASE TYPE N FOR NUMERIC OR A FOR ALPHANUMERIC
:**A
PLEASE TYPE FIELD KEY NO. IN THE RANGE 0 TO 9
0 DENOTES A NON-KEY FIELD, 1 DENOTES THE MAJOR KEY
:**0
CAN FIELD BE TREATED AS A DOMAIN? TYPE Y FOR YES OR N FOR NO
:**Y
PLEASE TYPE UP TO 70 CHARACTERS OF DESCRIPTION ABOUT THE FIELD
**DEPARTMENT MANAGER
CATALOGUE UPDATED ON 21/02/78 AT 20/17/44
DO YOU WISH TO ADD A NEW FILE DESCRIPTION OR DELETE AN EXISTING ONE?
PLEASE TYPE ADD OR DELETE, AT END TYPE FINISH.
:**FINISH
Contents of Catalogue

1,12, CATA PASS, 4, 21/02/79, 20/17/44,
DEPARTMENT, 4, DEPT1, DEPT2, DEPT3, DEPT4
DEPARTMENT DESCRIPTION RELATION
MT, 396, 66, DEPARTMENT, S, 1,
DEPT-NO, 4, 5, 2, N, L0, 1, A,
DEPARTMENT NUMBER
DEPT-NAME, 4, 7, 20, A, 0, Y,
DEPARTMENT NAME
DEPT-LOC, 4, 27, 20, A, 0, N,
DEPARTMENT LOCATION
DEPT-MANAGER, 4, 47, 20, A, 0, Y,
DEPARTMENT MANAGER
APPENDIX X

LISTING OF GENERATED COBOL PROGRAM

Pages 2 to 9 of this appendix list the COBOL program statements which would be generated to solve the model problem described in Section 5.9 of the main text of this thesis. Page 10 contains a sample of program output.

A vertical line at the left-hand side of certain listing lines is used to indicate statements which come from 'grown' macros. Intermediate blank lines in the program listing are not significant.
IDENTIFICATION DIVISION.
PROGRAM-ID. COBLO1.
DATE-WRITTEN. 09/12/77.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ICL-1907.
OBJECT-COMPUTER. ICL-1907 MEMORY SIZE 19000 WORDS.
SPECIAL NAMES.
*DATE IS Z-DATE *TIME IS Z-TIME.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT Z-REPORT-FILE ASSIGN TO PRINTER 1.
  SELECT Z-FILE-1 ASSIGN TO TAPES 1.
  SELECT Z-FILE-2 ASSIGN TO TAPES 2.
  SELECT Z-FILE-3 ASSIGN TO CARD-READER 1.
DATA DIVISION.
FILE SECTION.
FD Z-REPORT-FILE
  LABEL RECORDS OMITTED
  DATA RECORD IS Z-OUTREC.
  01 Z-OUTREC PICTURE X(70).
FD Z-FILE-1
  BLOCK CONTAINS 396 CHARACTERS
  RECORD CONTAINS 66 CHARACTERS
  LABEL RECORDS STANDARD VALUE OF ID "DEPARTMENT"
  DATA RECORD IS Z-INREC1.
  01 Z-INREC1.
    02 Z-IN-REC1-1.
      03 FILLER PICTURE X(4).
      03 DEPT-NO PICTURE 9(2).
      03 FILLER PICTURE X(40).
      03 DEPT-MANAGER PICTURE X(20).
FD Z-FILE-2
  BLOCK CONTAINS 490 CHARACTERS
  RECORD CONTAINS 70 CHARACTERS
  LABEL RECORDS STANDARD VALUE OF ID "PERSONNEL"
  DATA RECORD IS Z-INREC2.
  01 Z-INREC2.
    02 Z-IN-REC2-1.
      03 FILLER PICTURE X(4).
      03 EMPLOYEE-NO PICTURE 9(4).
      03 FILLER PICTURE X(52).
      03 JOIN-YEAR PICTURE 9(2).
      03 JOIN-MONTH PICTURE 9(2).
      03 JOIN-DAY PICTURE 9(2).
      03 FILLER PICTURE X(4).
FD Z-FILE-3
BLOCK CONTAINS 80 CHARACTERS
RECORD CONTAINS 80 CHARACTERS
LABEL RECORDS OMITTED
DATA RECORD IS Z-INREC3.
01 Z-INREC3.
   02 Z-IN-REC3-1.
      03 DEPT-NO PICTURE X(2).
      03 EMPLOYEE-NO PICTURE X(4).
      03 EMPLOYEE-NAME PICTURE X(20).
      03 FILLER PICTURE X(54).
WORKING-STORAGE SECTION.
77 Z-EOF-1 PICTURE 9 VALUE 0 USAGE COMPUTATIONAL.
77 Z-EOF-2 PICTURE 9 VALUE 0 USAGE COMPUTATIONAL.
77 Z-EOF-3 PICTURE 9 VALUE 0 USAGE COMPUTATIONAL.
77 Z-EOF-COUNT PICTURE 9 VALUE 0 USAGE COMPUTATIONAL.
77 Z-PICTURE 9 USAGE COMPUTATIONAL.
77 Z-LINE-COUNT PICTURE 99 VALUE 0 USAGE COMPUTATIONAL.
77 Z-PAGE-COUNT PICTURE 99 VALUE 45 USAGE COMPUTATIONAL.
77 Z-LEVEL-1.
      03 MONTHS-SERVICE PICTURE S9(18)
      VALUE 0 USAGE COMPUTATIONAL.
      03 NO-EMPLOYEES PICTURE S9(18)
      VALUE 0 USAGE COMPUTATIONAL.
02 Z-LEVEL-0.
      03 MONTHS-SERVICE PICTURE S9(18)
      VALUE 0 USAGE COMPUTATIONAL.
      03 NO-EMPLOYEES PICTURE S9(18)
      VALUE 0 USAGE COMPUTATIONAL.
01 Z-LINE-COUNTS.
   02 Z-LC-1 PICTURE 99 VALUE 6 USAGE COMPUTATIONAL.
   02 Z-LC-2 PICTURE 99 VALUE 1 USAGE COMPUTATIONAL.
   01 FILLER REDEFINES Z-LINE-COUNTS.
   02 Z-LC PICTURE 99 OCCURS 2 TIMES USAGE COMPUTATIONAL.
01 Z-CONTROL-BREAK.
   02 DEPT-NO PICTURE 9(2).
01 Z-TITLE-1.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(38) VALUE " X.Y.Z. MANUFACTURING COMPANY LIMITED ".
01 Z-TITLE-2.
   02 FILLER PICTURE X(24) VALUE SPACES.
   02 FILLER PICTURE X(21) VALUE " LONG SERVICE REPORT ".
01 Z-TITLE-3.
   02 FILLER PICTURE X(28) VALUE SPACES.
   02 FILLER PICTURE X(14) VALUE " CONFIDENTIAL ".
01 Z-PAGE-1.
   02 FILLER PICTURE X(57) VALUE " LONG SERVICE REPORT FOR THE YEAR ENDING 31ST DECEMBER 19 ".
   02 FILLER PICTURE X(8) VALUE " 77 PAGE ".
   02 Z-PAGE-COUNT PICTURE ZZZ9.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z-HEAD1-1</td>
<td>Department No.</td>
</tr>
<tr>
<td>Z-HEAD1-2</td>
<td>Employee No. Employee Name Date of Joining Length</td>
</tr>
<tr>
<td>Z-HEAD1-3</td>
<td>Department No.</td>
</tr>
<tr>
<td>Z-DETAIL-1</td>
<td>Employee No. Employee Name Join Date Length of Service</td>
</tr>
<tr>
<td>Z-SUBTOTAL1-1</td>
<td>Department No.</td>
</tr>
<tr>
<td>Z-TOTAL-1</td>
<td>Company Total Employees Average Service Years Months</td>
</tr>
<tr>
<td>Z-TOTAL-2</td>
<td>Average Length of Service Years Months</td>
</tr>
</tbody>
</table>
PROCEDURE DIVISION.

Z-INITIAL SECTION.

Z-PARA-1.
OPEN INPUT
Z-FILE-1
Z-FILE-2
Z-FILE-3
OUTPUT Z-REPORT-FILE.
PERFORM Z-END-PAGE-COUNTS VARYING Z FROM 2 BY -1 UNTIL Z EQUALS 1.
PERFORM Z-START-LINE VARYING Z FROM 1 BY 1 UNTIL Z GREATER THAN 2.
ACCEPT ZDATE IN Z-TUPLE FROM Z-DATE.
ACCEPT ZTIME IN Z-TUPLE FROM Z-TIME.
PERFORM Z-READ-1.
PERFORM Z-READ-2.
PERFORM Z-READ-3.

Z-PARA-2.
PERFORM Z-FETCH-TUPLE. IF Z-IGNORE EQUALS 1 GO TO Z-PARA-2.
MOVE CORRESPONDING Z-TUPLE-N TO Z-TUPLE.
MOVE CORRESPONDING Z-TUPLE TO Z-CONTROL-BREAK.
PERFORM Z-PARA-TITLE.
PERFORM Z-PARA-PAGE.
Z-PROCESSING SECTION.

Z-PARA-3.

MOVE JOIN-DAY IN Z-TUPLE TO Z-JOIN-DAY IN JOIN-DATE IN Z-TUPLE.
MOVE JOIN-MONTH IN Z-TUPLE TO Z-JOIN-MONTH IN JOIN-DATE IN Z-TUPLE.
MOVE JOIN-YEAR IN Z-TUPLE TO Z-JOIN-YEAR IN JOIN-DATE IN Z-TUPLE.
SUBTRACT JOIN-YEAR IN Z-TUPLE FROM 77 GIVING WORK-ITEM IN Z-TUPLE.
MULTIPLY WORK-ITEM IN Z-TUPLE BY 12 GIVING MONTHS-SERVICE IN Z-TUPLE.
SUBTRACT JOIN-MONTH IN Z-TUPLE FROM 13 GIVING WORK-ITEM IN Z-TUPLE.
ADD WORK-ITEM IN Z-TUPLE, MONTHS-SERVICE IN Z-TUPLE GIVING MONTHS-SERVICE IN Z-TUPLE.
DIVIDE MONTHS-SERVICE IN Z-TUPLE BY 12 GIVING L-O-S-YEARS IN Z-TUPLE
REMAINDER L-O-S-MONTHS IN Z-TUPLE.
IF L-O-S-YEARS IN Z-TUPLE NOT LESS THAN 10
GO TO Z-PARA-4. GO TO Z-PARA-5.
Z-PARA-4.
PERFORM Z-PARA-ADD.
PERFORM Z-PARA-WRITE.
Z-PARA-5. PERFORM Z-FETCH-TUPLE.
IF Z-IGNORE EQUALS 1 GO TO Z-PARA-5.
PERFORM Z-SEQUENCE-BREAK. GO TO Z-PARA-3.
Z-FINAL SECTION.

Z-PARA-FINAL.
PERFORM Z-TOTAL1.
IF Z-LINE-COUNT GREATER THAN 43 PERFORM Z-PARA-PAGE.
DIVIDE MONTHS-SERVICE IN Z-LEVEL-0 BY NO-EMPLOYEES IN Z-LEVEL-0 GIVING WORK-ITEM IN Z-TUPLE.
DIVIDE WORK-ITEM IN Z-TUPLE BY 12 GIVING AV-SER-YEARS IN Z-TUPLE
REMAINDER AV-SER-MONTHS IN Z-TUPLE.
PERFORM Z-TOTAL0-WRITE.
Z-CLOSE-DOWN. CLOSE
Z-FILE-1
Z-FILE-2
Z-FILE-3
Z-REPORT-FILE. STOP RUN.
Z-OTHER-PROCEDURES SECTION.
Z-PARA-ADD. ADD CORRESPONDING Z-TUPLE TO Z-LEVEL-1.
Z-PARA-WRITE.
IF Z-LINE-COUNT GREATER THAN Z-L-C(2) PERFORM Z-PARA-PAGE.
MOVE CORRESPONDING Z-TUPLE TO Z-DETAIL-1.
ADD 1 TO Z-ADV-LINES.
WRITE Z-OUTREC FROM Z-DETAIL-1 AFTER ADVANCING Z-ADV-LINES.
MOVE 0 TO Z-ADV-LINES.
ADD 1 TO Z-LINE-COUNT.
MOVE 0 TO Z-PAGE-SWITCH.
Z-END-PAGE-COUNTS. COMPUTE ZI = Z - 1.
ADD Z-L-C(Z) TO Z-L-C(Z1).


Z-HEAD1.
IF Z-LINE-COUNT GREATER THAN Z-L-C(1) PERFORM Z-PARA-PAGE
GO TO Z-HEAD1-EXIT.
IF Z-PAGE-SWITCH EQUALS 1 GO TO Z-HEAD1-EXIT.
PERFORM Z-HEAD1-WRITE.

Z-HEAD1-EXIT. EXIT.

Z-TOTAL1.
IF Z-LINE-COUNT GREATER THAN 42 PERFORM Z-PARA-PAGE.
DIVIDE MONTHS-SERVICE IN Z-LEVEL-1 BY
NO-EMPLOYEES IN Z-LEVEL-1 GIVING
WORK-ITEM IN Z-TUPLE.
DIVIDE WORK-ITEM IN Z-TUPLE BY
12 GIVING
AV-SER-YEARS IN Z-TUPLE
REMAINER AV-SER-MONTHS IN Z-TUPLE.
PERFORM Z-TOTAL1-WRITE.
MOVE 0 TO Z-PAGE-SWITCH.
ADD CORRESPONDING Z-LEVEL-1 TO Z-LEVEL-0.
SUBTRACT CORRESPONDING Z-LEVEL-1 FROM Z-LEVEL-1.

Z-PARA-TITLE.
MOVE SPACES TO Z-OUTREC.
WRITE Z-OUTREC BEFORE ADVANCING CHANNEL-1.
WRITE Z-OUTREC FROM Z-TITLE-1 AFTER ADVANCING 12 LINES.
WRITE Z-OUTREC FROM Z-TITLE-2 AFTER ADVANCING 2 LINES.
WRITE Z-OUTREC FROM Z-TITLE-3 AFTER ADVANCING 2 LINES.

Z-PARA-PAGE.
MOVE SPACES TO Z-OUTREC. ADD 1 TO Z-PAGE-COUNT IN Z-TUPLE.
MOVE CORRESPONDING Z-TUPLE TO Z-PAGE-1.
WRITE Z-OUTREC FROM Z-PAGE-1 AFTER ADVANCING 2 LINES.
MOVE 1 TO Z-ADV-LINES.
MOVE 4 TO Z-LINE-COUNT.
MOVE 1 TO Z-PAGE-SWITCH.
PERFORM Z-HEAD1-WRITE.

Z-HEAD1-WRITE.
ADD 2 TO Z-ADV-LINES.
WRITE Z-OUTREC FROM Z-HEAD1-1 AFTER ADVANCING Z-ADV-LINES.
WRITE Z-OUTREC FROM Z-HEAD1-2 AFTER ADVANCING 2 LINES.
WRITE Z-OUTREC FROM Z-HEAD1-3 AFTER ADVANCING 1 LINES.
MOVE 1 TO Z-ADV-LINES.
ADD 6 TO Z-LINE-COUNT.

Z-TOTAL1-WRITE.
ADD 2 TO Z-ADV-LINES.
MOVE CORRESPONDING Z-LEVEL-1 TO Z-SUBTOTAL1-1.
MOVE DEPT-NO IN Z-TUPLE TO DEPT-NO IN
Z-SUBTOTAL1-1.
MOVE AV-SER-YEARS IN Z-TUPLE TO AV-SER-YEARS IN
Z-SUBTOTAL1-1.
MOVE AV-SER-MONTHS IN Z-TUPLE TO AV-SER-MONTHS IN
Z-SUBTOTAL1-1.
WRITE Z-OUTREC FROM Z-SUBTOTAL1-1 AFTER Z-ADV-LINES.
MOVE 1 TO Z-ADV-LINES.
ADD 3 TO Z-LINE-COUNT.
Z-TOTAL0-WRITE.
ADD 2 TO Z-ADV-LINES.
MOVE CORRESPONDING Z-LEVEL-0 TO Z-TOTAL-1.
WRITE Z-OUTREC FROM Z-TOTAL-1 AFTER ADVANCING Z-ADV-LINES.
MOVE CORRESPONDING Z-TUPLE TO Z-TOTAL-2.
WRITE Z-OUTREC FROM Z-TOTAL-2 AFTER ADVANCING 1 LINES.

Z-READ-1 SECTION.
Z-READ-1-1. READ Z-FILE-1 AT END GO TO Z-READ-1-2.
GO TO Z-READ-1-EXIT.
Z-READ-1-2. MOVE 1 TO Z-EOF-1.
Z-READ-1-EXIT. EXIT.

Z-READ-2 SECTION.
Z-READ-2-1. READ Z-FILE-2 AT END GO TO Z-READ-2-2.
GO TO Z-READ-2-EXIT.
Z-READ-2-2. MOVE 1 TO Z-EOF-2.
Z-READ-2-EXIT. EXIT.

Z-READ-3 SECTION.
Z-READ-3-1. READ Z-FILE-3 AT END GO TO Z-READ-3-2.
GO TO Z-READ-3-EXIT.
Z-READ-3-2. MOVE 1 Z-EOF-3.
ADD 1 TO Z-EOF-COUNT.
Z-READ-3-EXIT. EXIT.

Z-SEQUENCE-BREAK SECTION.
Z-BREAK-1.
IF DEPT-NO IN Z-TUPLE-N EQUALS DEPT-NO IN Z-CONTROL-BREAK GO TO Z-BREAK-1-EXIT.
PERFORM Z-TOTAL1.
MOVE CORRESPONDING Z-TUPLE-N TO Z-TUPLE.
MOVE CORRESPONDING Z-TUPLE TO Z-CONTROL-BREAK.
PERFORM Z-HEAD1 THRU Z-HEAD1-EXIT.
GO TO Z-BREAK-EXIT.
Z-BREAK-1-EXIT. EXIT.
Z-BREAK-MOVE. MOVE CORRESPONDING Z-TUPLE-N TO Z-TUPLE.
Z-BREAK-EXIT. EXIT.

Z-SEQUENCE-BREAK SECTION.
Z-END-OF-DATA-TEST. IF Z-EOF-COUNT EQUALS 1 GO TO Z-FINAL.
MOVE 0 TO Z-IGNORE.

Z-SET-TUPLE-KEY.
MOVE DEPT-NO IN Z-INREC3 TO DEPT-NO IN Z-TUPLE-N.
MOVE EMPLOYEE-NO IN Z-INREC3 TO EMPLOYEE-NO IN Z-TUPLE-N.

Z-SET-TUPLE-DATA-1.
IF Z-EOF-1 EQUALS 1 GO TO Z-OPTION-1.

Z-MATCH-1.
IF DEPT-NO IN Z-INREC1 GREATER THAN DEPT-NO IN Z-TUPLE-N GO TO Z-OPTION-1,
IF DEPT-NO IN Z-INREC1 LESS THAN DEPT-NO IN Z-TUPLE-N GO TO Z-FETCH-1.
MOVE DEPT-MANAGER IN Z-INREC1 TO DEPT-MANAGER IN Z-TUPLE-N.
GO TO Z-MOVE-1-EXIT.

Z-FETCH-1. PERFORM Z-READ-1.
IF Z-EOF-1 EQUALS 0 GO TO Z-MATCH-1.
CLOSE Z-FILE-1 RETAIN OPEN INPUT Z-FILE-1 MOVE 0 TO Z-EOF-1.
PERFORM Z-READ-1.
Z-OPTION-1.
    MOVE 1 TO Z-IGNORE.
Z-MOVE-1-EXIT. EXIT.
Z-SET-TUPLE-DATA-2.
    IF Z-EOF-2 EQUALS 1 GO TO Z-OPTION-2.
Z-MATCH-2.
    IF EMPLOYEE-NO IN Z-INREC2 GREATER THAN
        EMPLOYEE-NO IN Z-TUPLE-N GO TO Z-OPTION-2.
    IF EMPLOYEE-NO IN Z-INREC2 LESS THAN
        EMPLOYEE-NO IN Z-TUPLE-N GO TO Z-FETCH-2.
    MOVE JOIN-YEAR IN Z-INREC2 TO JOIN-YEAR
        IN Z-TUPLE-N.
    MOVE JOIN-MONTH IN Z-INREC2 TO JOIN-MONTH
        IN Z-TUPLE-N.
    MOVE JOIN-DAY IN Z-INREC2 TO JOIN-DAY
        IN Z-TUPLE-N.
    GO TO Z-MOVE-2-EXIT.
    IF Z-EOF-2 EQUALS 0 GO TO Z-MATCH-2.
    CLOSE Z-FILE-2 RETAIN OPEN INPUT Z-FILE-2 MOVE 0 TO Z-EOF-2.
    PERFORM Z-READ-2.
Z-OPTION-2.
    MOVE 1 TO Z-IGNORE.
Z-MOVE-2-EXIT. EXIT.
Z-SET-TUPLE-DATA-3.
    MOVE EMPLOYEE-NAME IN Z-INREC3 TO EMPLOYEE-NAME
        IN Z-TUPLE-N.
    PERFORM Z-READ-3.

****
LONG SERVICE REPORT FOR THE YEAR ENDING 31ST DECEMBER 1977 PAGE 3

DEPARTMENT NO. 10 DEPARTMENT MANAGER F.G. PAYTON-MCDOWELL

<table>
<thead>
<tr>
<th>EMPLOYEE NO.</th>
<th>EMPLOYEE NAME</th>
<th>DATE OF JOINING</th>
<th>LENGTH OF SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1340</td>
<td>F.G. PAYTON-MCDOWELL</td>
<td>16 04 62</td>
<td>15 9</td>
</tr>
<tr>
<td>550</td>
<td>J. METCALFE</td>
<td>21 01 66</td>
<td>12 0</td>
</tr>
<tr>
<td>550</td>
<td>D.M. MIDLANE</td>
<td>13 12 66</td>
<td>11 1</td>
</tr>
</tbody>
</table>

DEPARTMENT NO. 10 NO. OF EMPLOYEES 3 AVERAGE SERVICE 12 11

DEPARTMENT NO. 12 DEPARTMENT MANAGER C.W. RUSSELL

<table>
<thead>
<tr>
<th>EMPLOYEE NO.</th>
<th>EMPLOYEE NAME</th>
<th>DATE OF JOINING</th>
<th>LENGTH OF SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>620</td>
<td>C.W. RUSSELL</td>
<td>18 06 62</td>
<td>15 7</td>
</tr>
</tbody>
</table>

DEPARTMENT NO. 12 NO. OF EMPLOYEES 1 AVERAGE SERVICE 15 7

DEPARTMENT NO. 14 DEPARTMENT MANAGER B.J.S. MOORE

<table>
<thead>
<tr>
<th>EMPLOYEE NO.</th>
<th>EMPLOYEE NAME</th>
<th>DATE OF JOINING</th>
<th>LENGTH OF SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>910</td>
<td>B.J.S. MOORE</td>
<td>20 04 61</td>
<td>16 9</td>
</tr>
</tbody>
</table>

DEPARTMENT NO. 14 NO. OF EMPLOYEES 1 AVERAGE SERVICE 16 9

COMPANY TOTAL 15 EMPLOYEES
AVERAGE LENGTH OF SERVICE 14 YEARS 6 MONTHS
22. DIALS AND STAGE5 SUBFILES

1. WHAT IS STAGE 5?

STAGE 5 is the process of creating new data items for the database. Although your database may be extended on a permanent basis in consultation with the database administrator, you may create new data items which are available only during this system session. The processing of these temporary items will be entirely under your control and later you will be given an opportunity to specify your processing requirements.

2. INTERMEDIATE RESULTS OF CALCULATIONS OR FOR REMAINDERS ARISING FROM ARITHMETIC OPERATIONS

Your database may also be temporarily extended by the inclusion of a compound domain, formed by concatenating the items from two or more other domains within the database. E.g. the temporary domain address may be compounded from the existing domain STREET, TOWN and POSTCODE. Compound domains are always of alphanumeric type.

3. CREATE ANY ITEMS REQUIRED FOR HOLDING INTERMEDIATE RESULTS OF CALCULATIONS OR FOR REMAINDERS ARISING FROM ARITHMETIC OPERATIONS

CREATE ANY ITEMS REQUIRED FOR HOLDING INTERMEDIATE RESULTS OF CALCULATIONS OR FOR REMAINDERS ARISING FROM ARITHMETIC OPERATIONS.

4. NOTE: SET CHECK POINT 5

NOTE: SET CHECK POINT 5.

5. E.G. THE TEMPORARY DOMAIN ADDRESS MAY BE COMPOUNDED FROM THE EXISTING DOMAINS STREET, TOWN AND POSTCODE. COMPOUND DOMAINS ARE ALWAYS OF ALPHANUMERIC TYPE.

6. INPUT

INPUT

7. DEF STAGES5

DEF STAGES5.

8. EXTENSION OF DATABASE - CHECK POINT 5

EXTENSION OF DATABASE - CHECK POINT 5.

9. NOTE: SET CHECK POINT 5.

NOTE: SET CHECK POINT 5.

10. SELECT DIALS

SELECT DIALS.

11. #500 READ DLINE & S01

#500 READ DLINE & S01.

12. #501 DEPOSIT & S01

#501 DEPOSIT & S01.

13. NOTE: SELECT DIALS LIST

NOTE: SELECT DIALS LIST.

14. READ DLINE & S01

READ DLINE & S01.

15. DEPOSIT & S01

DEPOSIT & S01.

16. NOTE: SELECT DIALS LIST

NOTE: SELECT DIALS LIST.

17. READ DLINE & S01

READ DLINE & S01.

18. DEPOSIT & S01

DEPOSIT & S01.

19. NOTE: SELECT DIALS LIST

NOTE: SELECT DIALS LIST.

20. READ DLINE & S01

READ DLINE & S01.

21. DEPOSIT & S01

DEPOSIT & S01.

22. NOTE: SELECT DIALS LIST

NOTE: SELECT DIALS LIST.

23. READ DLINE & S01

READ DLINE & S01.

24. DEPOSIT & S01

DEPOSIT & S01.

25. NOTE: SELECT DIALS LIST

NOTE: SELECT DIALS LIST.

26. READ DLINE & S01

READ DLINE & S01.

27. DEPOSIT & S01

DEPOSIT & S01.

28. NOTE: SELECT DIALS LIST

NOTE: SELECT DIALS LIST.

29. READ DLINE & S01

READ DLINE & S01.

30. DEPOSIT & S01

DEPOSIT & S01.
IF #GSTC.EQ.1,GOTO 6
CALL FST(\#LV1,1,1,1)
IF #LV1.IN.\#GOTO 4
IF #LV1.EQ.\#GOTO 5
NOTE: INVALID RESPONSE
CALL 2100,7

NOTE: OFFER USERS OPTION TO CREATE ALPHANUMERIC ITEMS
DEPOSITO BOX OR VISITATORY PRIVATE TEMPORARY ITEMS WHICH CONTAINS
WRITE
DEPOSITO ALPHANUMERIC DATAS
WRITE
CALL 2200
NOTE: READ AND VALIDATE USERS REPLY
CALL 9000

SQUASH
IF #GSCC.EQ.1,GOTO 6
CALL FST(\#LV1,1,1,1)
IF #LV1.EQ.1,GOTO 10
IF #LV1.EQ.\#GOTO 26
NOTE: INVALID RESPONSE
CALL 2100,9

NOTE: REQUEST PM AND GENERATE ITEM NAME,
DEPOSITO PLEASE TYPE THE NAME OF THE ALPHANUMERIC ITEMS
WRITE
CALL 9000
WRITE #S09
#LENGTH=15
CALL 200
IF #LENGTH.EQ.0,GOTO 11
NOTE: INVALID NAME
DEPOSITO #S09 IS AN INVALID NAME, FOR REASON LISTED ABOVE
WRITE
GOTO 12

NOTE: BINARY SEARCH DOMAIN INDEX FOR DUPLICATE NAME
CALL 9000
IF #LENGTH.EQ.0,GOTO 13
DEPOSITO DOMAIN #S09 ALREADY EXISTS, PLEASE THINK OF A NEW NAME.
WRITE
GOTO 12

NOTE: REQUEST SIZE INFORMATION
DEPOSITO HOW MANY CHARACTERS DOES ITEM #S09 CONTAIN?
WRITE
DEPOSITO THE DEFAULT REPLY IS 30.
WRITE
NOTE: READ AND VALIDATE SIZE
CALL 9000
SQUASH
IF #GSCC.LT.1,GOTO 16
IF #GSCC.GT.2,GOTO 12
#GROUP=#GSCC
CALL FST(\#LV1,1,1,1)
IF #GSCC.EQ.0,GOTO 14
IF #GROUP.EQ.1,GOTO 15
CALL FST(\#S09,\#GROUP,\#LENGTH,1,1)
#GROUP=1
CALLCOV(\#LSIZE,\#S09)
IF #LENGTH.LT.3,GOTO 14
IF #LENGTH.GT.5,GOTO 14
GOTO 16
NOTE: INVALID SIZE
WRITE
DEPOSITO SIZE INVALID, MUST LIE IN THE RANGE 1 TO 58
LSIZE=30
NOTE: REQUEST INITIAL VALUE

DEPOSITO DON'T YOU DISPER TO ENTER AN INITIAL VALUE IN %S0973

WRITE

CALL 2200

DEPOSITO BY DEFAULT THE ITEM WILL INITIALLY CONTAIN BLANKS, S

WRITE

NOTE: READ AND VALIDATE USERS RESPONSE

NOTE: REQUEST INITIAL VALUE

DEPOSITO PLEASE ENTER THE INITIAL VALUE FOR %S0973 THE CHARACTERS

WRITE

DEPOSITO YOU TYPE WILL BE PLACED LEFT JUSTIFIED IN THE ITEM AND

WRITE

DEPOSITO PADDED OUT WITH BLANKS, EXCESS CHARACTERS WILL BE IGNORES

CALL COPY (57, D.)

WRITE

CALL 9000

GROUP = LSIZE

VSTR1 A, 0

GOTO 23

NOTE: INITIAL VALUE BLANK

RESET #501

DEPOSITO .

NOTE: ENTER THE MAIN INDEX AND UPDATE DOMAIN COUNT

RESET #503

DEPOSITO #509, E, LSIZE, A, 0, S

INSERT ALLOW, #502

#GOCT = GOCT

NOTE: GENERATE CODE FOR DATA DESCRIPTION

LABEL OFF

LABEL 02 #509 PICTURE X(LSIZE) VALUE

LABEL 02

NOTE: CREAT ALL ITEMS TO BE CREATED

DEPOSITO HAVE YOU ALPHANUMERIC ITEMS TO CREATE?

GOTO 33

NOTE: VALIDATE NAME STARTING IN 1ST POSITION OF INPUT BUFFER

NOTE: LUMNER = ERROR INDICATOR - INITIALLY CLEARED TO ZERO

Write #501

NOTE: TEMPORARILY SAVE CONTENTS OF INPUT BUFFER

NOTE: CHECK NO. OF CHARACTERS KEYED

IF #GSTC GT #LENGTH GOTO 205

#GSTATE FOR IMBEDDED BLANKS

IF #S01 HE #S02 GOTO 203

CALL CHECK FIRST CHARACTER FOR HYPHEN OR Z

IF #PLV1 IN 'I-V' GOTO 264

NOTE: CHECK LAST CHARACTER FOR HYPHEN

#GSTATE = #GSTC

CALL #PLV1, V1, I-7, GOTO 205

IF #PLV1 IN 'A-Z', GOTO 205

NOTE: SET UP LOOP TO CHECK FOR ALPHABETIC, NUMERIC AND HYPHEN ONLY

LABEL #501

NOTE: ZEROISE COUNT OF ALPHA CHARACTERS

LABEL #501

NOTE: TEST NUMERIC

NOTE: CHECK LAST CHARACTER
TESTDIGT(LV)  200
IF #GOST, EQ 0, GOTO 206
NOTE: TEST ALPHABETIC
TESTLET(LV)  201
IF #GOST, EQ 0, GOTO 207
NOTE: TEST HYPERH
NOTE: INVALID CHARACTER FOUND
DEPOSITO #LV%INVALID%CHARACTER.%ONLY%A%TO%Z%AND%-%ALLOWED
CALCOPY(54,D)
WRITE
NOTE: SET ERROR INDICATOR ON
#LNER=1
NOTE: UPDATE POINTER AND TEST FOR END OF LOOP
#GTOP=LV
IF #GTOP, GT, #GOST, GOTO 214
GOTO 213
NOTE: TEST FOR AT LEAST ONE ALPHABETIC CHARACTER
NOTE: NO ALPHABETIC CHARACTERS PRESENT
DEPOSITO AT%LEAST%ONE%ALPHABETIC%CHARACTER%MUST%BE%INCLUDED
WRITE
NOTE: NAME TOO LONG
DEPOSITO NOT%THREE%ALPHABETIC%CHARACTERS%IN%NAME
WRITE
#LNER=1
GOTO 200
NOTE: BLANK%NOT%ALLOWED
DEPOSITO NOT%BLANK%NOT%ALLOWED
WRITE
#LNER=1
GOTO 210
NOTE: FIRST CHARACTER HYPERPH OR 2
DEPOSITO 2%OR%HYPERPH%NOT%ALLOWED%AS%FIRST%CHARACTERS
WRITE
#LNER=1
GOTO 211
NOTE: LAST CHARACTER HYPERPH
DEPOSITO HYPERPH%NOT%ALLOWED%AS%LAST%CHARACTERS
WRITE
#LNER=1
GOTO 212
NOTE: ADD 1 TO ALPHABETIC COUNT
ALPHABET=1
GOTO 200
NOTE: ROUTINE TO SEARCH DOMAIN INDEX FOR THE DOMAIN NAME CONTAINED
IN #GSO
NOTE: DOMAIN COUNT = #INDCT
NOTE: #LIND=%INDICATOR IF FOUND 1=NOT FOUND
0000 #LIND=0
SELECT DOMAIN INDEX
#LIND=1
GOTO 200
0005 IF #LOW, GT, #HIGH, GOTO 3001
#LLOW=LOW48HIGH
#HIGH=HIGH48LOW
FREAD #LLOW, #HIGH
NOTE: EXTRACT DOMAIN NAME
VSTR #SO1, 1, #SO1, GOTO 801
IF #SO1, LT, #SO1, GOTO 801
264 28004 EXIT
265 28001 #LIND=1
569 26004 GOTO 28001
567 26002 #LHIGH=#LLOW+1
565 26000 GOTO 28000
WRITE DEPOSIT OF ANY DIGITS DOES NOT CONSIST?
WRITE DEPOSIT THE DEFAULT REPLIES IS 18.
WRITE NOTE: READ AND VALIDATE SIZE
CALL 9000
SQUASH
IF POSTC.LT.1, GOTO 34
IF #POSTC.GT.2, GOTO 35
#GROUP=1
#GROUP=#POSTC.
CALLSTR(#LV1, C,)
TESTDIGIT(#LV1)
IF #POSTC.LE.0, GOTO 35
#GROUP=GROUP+1
IF #POSTC.LE.0, GOTO 36
CALLSTR(#L_SIZE, L_SIZE)
#GROUP=1
CALLCCV(#L_SIZE, L_SIZE)
IF #POSTC.LE.0, GOTO 35
WRITE DEPOSIT SIZE IS INVALID. IT MUST LIE IN THE RANGE 1 TO 16.
WRITE GOTO 33.
NOTE: REQUEST DECIMAL POINT LOCATION
WRITE DEPOSIT DO YOU WISH TO SPECIFY A DECIMAL POINT LOCATION OTHERS
WRITE DEPOSIT THANK AFTER THE RIGHT HOST DIGIT.
WRITE CALL 2200
CALL 9000
SQUASH
CALLSTR(#LV1, 1, 1)
IF #LV1 IN, '0', GOTO 40
IF #LV1 EQ, 'Y', GOTO 44
NOTE: INVALID RESPONSE
OBEY 2100, 38
NOTE: SET DEFAULT POINT LOCATION VALUES
NOTE: REQUEST DECIMAL POINT LOCATION
WRITE DEPOSIT PLEASE SPECIFY FOR LEFT OR RIGHT, THE DEFAULT REPLIES
WRITE CALLCOPY(57, LASTDIGIT)
WRITE DEPOSIT TO WRITE
WRITE CALLCOPY(57, RIGHT)
WRITE NOTE: READ AND VALIDATE USERS REPLY
CALL 9000
SQUASH
IF #POSTC.GT.1, GOTO 42
CALLSTR(#LDIR, 1, 1)
IF #LDIR IN, 'R', GOTO 43
NOTE: INVALID RESPONSE
OBEY 2100, 45
ALDIR='R'
ALWORD='RIGHT'
ALWORD='LEFT'
NOTE: REQUEST DECIMAL POINT LOCATION
WRITE DEPOSIT PLEASE SPECIFY FOR LEFT OR RIGHT, THE DEFAULT REPLIES
WRITE CALLCOPY(57, LASTDIGIT)
WRITE DEPOSIT TO WRITE
WRITE CALLCOPY(57, RIGHT)
WRITE NOTE: READ AND VALIDATE USERS REPLY
CALL 9000
SQUASH
IF #POSTC.GT.1, GOTO 42
CALLSTR(#LDIR, 1, 1)
IF #LDIR IN, 'R', GOTO 43
NOTE: INVALID RESPONSE
OBEY 2100, 45
ALDIR='R'
ALWORD='RIGHT'
ALWORD='LEFT'
NOTE: REQUEST DECIMAL POINT LOCATION
WRITE DEPOSIT PLEASE SPECIFY FOR LEFT OR RIGHT, THE DEFAULT REPLIES
WRITE CALLCOPY(57, LASTDIGIT)
WRITE DEPOSIT TO WRITE
WRITE CALLCOPY(57, RIGHT)
WRITE NOTE: READ AND VALIDATE USERS REPLY
CALL 9000
SQUASH
IF #POSTC.GT.1, GOTO 42
CALLSTR(#LDIR, 1, 1)
IF #LDIR IN, 'R', GOTO 43
NOTE: INVALID RESPONSE
OBEY 2100, 45
ALDIR='R'
ALWORD='RIGHT'
ALWORD='LEFT'
NOTE: REQUEST DECIMAL POINT LOCATION
WRITE DEPOSIT PLEASE SPECIFY FOR LEFT OR RIGHT, THE DEFAULT REPLIES
WRITE CALLCOPY(57, LASTDIGIT)
WRITE DEPOSIT TO WRITE
WRITE CALLCOPY(57, RIGHT)
WRITE NOTE: READ AND VALIDATE USERS REPLY
CALL 9000
SQUASH
IF #POSTC.GT.1, GOTO 42
CALLSTR(#LDIR, 1, 1)
IF #LDIR IN, 'R', GOTO 43
NOTE: INVALID RESPONSE
OBEY 2100, 45
ALDIR='R'
ALWORD='RIGHT'
ALWORD='LEFT'
WHITE
WHITE

CALL 9000
GOTO 8100

IF #GSTC EQ #GINP GOTO 8101
IF #GINP NE #GSTC GOTO 8102
GOTO 8106

IF #GINP LE #GSTC GOTO 8103
IF #GINP LE #GSTC GOTO 8108

CALL STR(#{LV1:1}, 1)
TESTDIGT(#{LV1})
GOTO 8107

! NOTE: ROUTINE TO SEARCH DOMAINDX FOR THE DOMAIN NAME CONTAINED
! NOTE: IN #GSP
! NOTE: #DOMAINDX = #GDCUT
! #IND = INDICATOR 0#FOUND 1=NOT FOUND

#IND=0
SELECT DOMAINDX

#DOMAINDX
GOTO 8000

#DOMAINDX
#GDCUT
#CUR=1
#LH=1
GOTO 8001
GOTO 8004
GOTO 8005
GOTO 8006
GOTO 8007
GOTO 8008
GOTO 8009
GOTO 8010
GOTO 8011
GOTO 8012
GOTO 8013

! NOTE: EXTRACT DOMAIN NAME
! #LH=1
! #DOMAINDX=1
! #DOMAINDX=#CUR+1
! #LH=#DOMAINDX
! #GDCUT=#DOMAINDX
! #DOMAINDX=#DOMAINDX+1

! WRITE #GDCUT
WRITE #GDCUT:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
WRITE #DOMAINDX:
609 IF #GTS# EQ 0, GOTO 207
610 LISTI 0, GOTO 207
611 IF #LVL1# EQ 1, GOTO 207
612 NOTE: INVALID SYMBOL OR DIGIT FOUND
613 CALL COPY(54, D)
614 WRITE
615 NOTE: SET ERROR INDICATOR ON
616 #LNERR=1
617 NEXT
618 NOTE: UPDATE POINTER AND TEST FOR END OF LOOP
619 IF #GTE# GT #LSC# GOTO 214
620 GOTO Z
621 NOTE: TEST FOR AT LEAST ONE ALPHABETIC CHARACTER
622 IF #LHL# EQ #LAI# GOTO 214
623 NOTE: NO ALPHABETIC CHARACTER PRESENT
624 CALL COPY AT LEAST ONE ALPHABETIC CHARACTER MUST BE INCLUDED
625 WRITE
626 #LNERR=1
627 GOTO 208
628 NOTE: NAME TOO LONG
629 DEPOSIT INTO NAME\%ALPHABETIC\%CHARACTERS\%IN\%NAMES
630 WRITE
631 #LNERR=1
632 GOTO 209
633 NOTE: BLANKS NOT ALLOWED
634 DEPOSIT INTO BLANKS NOT ALLOWED
635 WRITE
636 #LNERR=1
637 GOTO 210
638 NOTE: NAME TOO LONG
639 DEPOSIT INTO NAME\%ALPHABETIC\%CHARACTERS\%IN\%NAMES
640 WRITE
641 #LNERR=1
642 GOTO 211
643 NOTE: LAST CHARACTER MUST BE ALPHABETIC CHARACTER
644 DEPOSIT INTO LAST CHARACTER MUST BE ALPHABETIC CHARACTER
645 WRITE
646 #LNERR=1
647 GOTO 212
648 NOTE: NAME TOO LONG
649 DEPOSIT INTO NAME\%ALPHABETIC\%CHARACTERS\%IN\%NAMES
650 WRITE
651 #LNERR=1
652 GOTO 206
653 NOTE: NAME TOO LONG
654 DEPOSIT INTO NAME\%ALPHABETIC\%CHARACTERS\%IN\%NAMES
655 WRITE
656 #LNERR=1
657 GOTO 999
658 NOTE: CHAIN TO NEXT SECTION OF STAGE
659 CONTINUE STAGE5=211
660 GOTO 999
661 NOTE: PRINT INVALID RESPONSE MESSAGE
662 DEPOSIT INTO INVALID RESPONSE MESSAGE
663 WRITE
664 RETURN
665 NOTE: PLEASE TYPE A RESPONSE AND DETECT A PLEA FOR HELP
666 DEPOSIT INTO PLEASETYPE\%FORYES\%FORNO\%THEDEFAULT\%REPLY\%IS\%YES
667 CALL COPY(54, D)
668 WRITE
669 EXIT
670 NOTE: ROUTINE TO READ A RESPONSE AND DETECT A PLEA FOR HELP
671 CALL VSTR(1, LVL1, 1, 5)
672 IF LVL1 EQ 'HELP', GOTO 900;
673 EXIT
674 GOTO 999
675 CONTINUE
676 CALL (HELP1)
677 CONTINUE
678 CALL (HELP1)
DEPOSITO PLEASETYPE%THE%NAME%OF%THE%ALPHANUMERIC%DOMAIN%TO%BE%CONCATENATED%IN%ORDER%FROM%LEFT%TO%RIGHT.

WRITE

NOTE:  READ AND VALIDATE DOMAIN NAME
CALL 57
WRITE

NOTE:  WE WILL BE CONCATENATED IN ORDER FROM LEFT TO RIGHT.

WRITE

CALL COPY

2,7 AND VALIDATE DOMAIN NAME

NOTE:  ENTER DOMAIN IN INDEX AND UPDATE COUNT
GOTO 79

DEPOSITO HAVEYOU ANY%ALPHANUMERIC%DOMAINS%TO%INCLUDE%IN%THIS%CONCATENATION?

WRITE

NOTE:  ENTER ALPHANUMERIC%IN%TYPE.

WRITE

NOTE:  DEPOSITO HAVEYOU ANY%ALPHANUMERIC%CONCATENATION%TO%CREATE?

NOTE:  GENERATE FINAL STATEMENTS FOR CONCAT AND TEMPLATE MACROS

NOTE:  GENERATE FINAL STATEMENTS FOR CONCAT AND TEMPLATE MACROS

NOTE:  ENTER DOMAIN IN INDEX AND UPDATE COUNT
GOTO 79

DEPOSITO HAVEYOU ANY%ALPHANUMERIC%CONCATENATION%TO%CREATE?

NOTE:  ENTER DOMAIN IN INDEX AND UPDATE COUNT
GOTO 79

DEPOSITO HAVEYOU ANY%ALPHANUMERIC%CONCATENATION%TO%CREATE?
NOTE: SAVE PREVIOUSLY GENERATED TEMPITEM STATEMENTS

NOTE: GENERATE FIRST STATEMENTS FOR CONCAT MACRO

NOTE: OFFER USER OPTION TO CONCATENATE DATA ITEMS

SELECT DIAL5

FREAD #LINE=1 #S01

WRITE #LINE=#LINE+1

IF #LINE.LE.0, GO TO 94

DEPOSIT #RECOUNT (1) TO TEMPORARILY EXTEND YOUR DATABASE

WRITE

DEPOSIT 0 INCLUSION OF X% COMPOUND DOMAIN

WRITE

CALL 2200

CALL 9000 AND VALIDATE REPLY

SQUASH

IF #STC.GT.1, GOTO 71

CALL FILE (11111)

IF #LV.I/EQ.0, GOTO 90

IF #LV.I.EQ.1, GOTO 72

NOTE: INVALID RESPONSE

DEPOSIT REQUEST NAME

DEPOSIT PLEASE ENTER NAME OF THE COMPOUND DOMAIN WHICH YOU WIS

CALL COPY (17, SHX TO CREATE)

WRITE

NOTE: READ AND VALIDATE NAME

DEPOSIT IS AN/

NOTE: NAME FOR REASON LISTED ABOVE

SAVE INDEX- POINTER

WHILE PT = NULL

NOTE: INITIALIZE COUNT FOR CHARACTERS

WHILE CT = 0

NOTE: GENERATE LEVEL- 0 ENTRY

DECLARE #JOINTV (1) = 0

SAVE #TEMPITEM

NOTE: FOR CONSTITUENT DOMAIN ITEMS

#LW=#FIRST

DEPOSIT ITEMS FROM THE COMPOUND WHICH YOU ARE ABOUT TO SPECIFY.
1. **NOTE**: **INVALID** RESPONSE

2. **OBEY** 2100, 50

3. **NOTE**: **TOTALS** NOT **REQUIRED**

4. **GOTO** 57

5. **NOTE**: **TOTALS** REQUIRED

6. **GOTO** 58

7. **RESET** #502

8. **DEPOSITO** #502, #L7, #LSIZE, n #DIR #LSHT, n, n

9. **INSERT** #L10, #502

10. **NOTE**: **GENERATE** LEVEL 02-ENTRY FOR-NUMERIC ITEM

11. **IF** #LSHT. EQ. C, GOTO 60

12. **IF** #DIR. EQ. 'L', GOTO 61

13. **IF** #L7. = 'P', GOTO 62

14. **IF** #LV4. EQ. #LSHT, GOTO 64

15. **LABELOFF**

16. **LABELON**

17. **GOTO** 65

18. **LABELOFF**

19. **LABELON**

20. **GOTO** 66

21. **LABELOFF**

22. **LABELON**

23. **GOTO** 67

24. **LABELOFF**

25. **LABELON**

26. **GOTO** 68

27. **LABELOFF**

28. **LABELON**

29. **GOTO** 69

30. **LABELOFF**

31. **LABELON**

32. **GOTO** 70

33. **LABELOFF**

34. **LABELON**

35. **GOTO** 71

36. **LABELOFF**

37. **LABELON**

38. **GOTO** 72

39. **LABELOFF**

40. **LABELON**

41. **GOTO** 73

42. **LABELOFF**

43. **LABELON**

44. **GOTO** 74

45. **LABELOFF**

46. **LABELON**

47. **GOTO** 75

48. **LABELOFF**

49. **LABELON**

50. **GOTO** 76

51. **LABELOFF**

52. **LABELON**

53. **GOTO** 77

54. **LABELOFF**

55. **LABELON**

56. **GOTO** 78

57. **LABELOFF**

58. **LABELON**

59. **GOTO** 79

60. **LABELOFF**

61. **LABELON**

62. **GOTO** 80

63. **LABELOFF**

64. **LABELON**

65. **GOTO** 81

66. **LABELOFF**

67. **LABELON**

68. **GOTO** 82

69. **LABELOFF**

70. **LABELON**

71. **GOTO** 83

72. **LABELOFF**

73. **LABELON**

74. **GOTO** 84

75. **LABELOFF**

76. **LABELON**

77. **GOTO** 85

78. **LABELOFF**

79. **LABELON**

80. **GOTO** 86

81. **LABELOFF**

82. **LABELON**

83. **GOTO** 87

84. **LABELOFF**

85. **LABELON**

86. **GOTO** 88

87. **LABELOFF**

88. **LABELON**

89. **GOTO** 89

90. **LABELOFF**

91. **LABELON**

92. **GOTO** 90

93. **LABELOFF**

94. **LABELON**

95. **GOTO** 91

96. **LABELOFF**

97. **LABELON**

98. **GOTO** 92

99. **LABELOFF**

100. **NOTE**: ROUTINE TO VALIDATE A NUMERIC LITERAL - MAX SIZE 60 CHAR.
NOTE: REQUEST DISPLACEMENT
CALLCOPY(57, AL, POINT, NX, MARCH)
WRITE
DEPOSIT TO THE\#1 Dir%, THE\#1\#host\%DIGIT, DEFAULT\%REPLY IS X%0$
NOTE: READ AND VALIDATE REPLY
CALL 9000
SQUASH
CALLSTR(11, GOTO, 48)
GOTO 48
IF #LSHF. EQ., "1", GOTO 57
TESTDIGIT(#LSHF)
GOTO 48
CALLCOPY(#LSHF, #LSHF)
GOTO 46
#LSHF=0
GOTO 46
NOTE: INVALID REPLY
DEPOSITI DISPLACEMENT\%MUST BE\%IN\%THE\%RANGE\%1\%TO\%9$
WRITE
GOTO 47
NOTE: REQUEST\#INITIAL\%VALUE
DEPOSIT DOXY\#HISP\#:ENTER\#AN\#INITIAL\%VALUE\%FOR\#S097$
WRITE
GOTO 2200
DEPOSIT DOXY\#DEFAULT\#:THE\#VALUE\#WILL\#INITIALLY\#CONTAIN\#ZERO\$.s
WRITE
NOTE: READ AND VALIDATE REPLY
CALL 9000
SQUASH
IF #GTC. GT, 1, GOTO 50
CALLSTR(11, LV1, 11)
GOTO 50
IF #LV1. IN. IN. "GOTO 51
IF #LV1. EQ. 1Y, GOTO 52
NOTE: INVALID RESPONSE
GOTO 2100
NOTE: 0\#DEFAULT VALUE OF ZERO SET
DEPOSITI OS
GOTO 58
NOTE: REQUEST\#INITIAL\%VALUE
DEPOSIT DOXY\#PLEASE\#:ENTER\#:THE\#INITIAL\#VALUE\#:FOR\#:S09\$.s
WRITE
DEPOSITI THE\#VALUE\#MUST\#CONSIST\#OF\#DIGITS\#AND\#MAY\#INCLUDE\#A\#DECIM
CALLCOPY(57, AL, POINT, XUNCH)
WRITE
DEPOSITI HAVE\#NOT\#:THE\#:LAST\#CHARACTER.\#THE\#SIGN\#:IF\#PRESENT.\#MUST
CALLCOPY(57, 1, THEFIRST)
WRITE
DEPOSITI CHARACTER.\#BY\#:DEFAULT\#:THE\#SIGN\#:IS\#POSITIVE\$.s
WRITE
NOTE: READ AND VALIDATE REPLY
CALL 8100
NOTE: STORE LITERAL
WRITE 2501
NOTE: REQUEST LITERAL
DEPOSITI DOXY\#:THE\#SYSTEM\%TOTAL\%ACCUMULATE\%\#TOTAL FOR\#T18$
CALLCOPY(55, LV1, 15)
WRITE
GOTO 2200
NOTE: READ AND VALIDATE USERS REPLY
CALL 9000
SQUASH
IF #GTC. GT. 1, GOTO 70
CALLSTR(11, LV1, 11)
IF #LV1. IN. IN. "GOTO 71

65 2 IF DROP HEL, GOTO 1
66 4 **NOTE: PRINT MESSAGE: DIALOGUE**
67 7 **WLINE = 74**
68 10 CALL 2000
69 13 **NOTE: READ AND VALIDATE RESPONSE**
70 16 **WLINE = 72**
71 19 CALL 2000
72 22 IF GTS GT, GOTO 11
73 25 CALL STRUFL(1, 1)
74 28 IF JVM1, INEOS!, GOTO 12
75 31 **NOTE: INVALID RESPONSE**
76 34 OSER #505
77 37 IF JVM1, EQ, 10, GOTO 13
78 40 **NOTE: SYSTEM MESSAGE**
79 43 **RES 15**
80 46 GOTO 15
81 49 **NOTE: REQUEST USERS MESSAGE**
82 52 DEPOSIT PLEASE TYPE YOUR MESSAGE, Character% in excess of % will be ignored.
86 74 **NOTE: ROUTINE TO PRINT DIALOGUE FROM DIAL6 FILE**
87 77 CALL 9000
88 80 WRITE, #505
89 90 **NOTE: ROUTINE TO PRINT DIALOGUE FROM DIAL6 FILE**
90 100 SELECT DIAL6
91 103 **WLINE = 501**
92 106 DEPOSITO #505
93 109 **WLINE = 501**
94 112 WRITE
95 115 IF JVM1, EQ, JVM2, GOTO 2001
96 118 **RATE: PRINT INVALID RESPONSE MESSAGE**
97 121 52000 DEPOSITO INVALID RESPONSE.
98 124 WRITE
99 127 RETURN
100 130 **NOTE: ROUTINE TO READ A RESPONSE AND DETECT A PLEA FOR HELP**
101 133 DEPOSITO $5
102 136 WRITE
103 139 READ 
104 142 CALLS, R(H, JVM1, 1, 2)
105 145 IF JVM1, EQ, JVM2, GOTO 9001
106 148 **NOTE: CHAIN TO STAGE7**
107 151 CONTINUE
108 154 **CHAIN, STAGE71**
109 157 **STAGE72**
110 160 **CONTINUE**
111 163 9990 CONTINUE
112 166 END
113 169 **INPUT**
114 172 **DIAL7**
115 175 7 - SELECTION OF DATA FOR RETRIEVAL
116 178 **YOU ARE ABOUT TO SPECIFY THE CONDITION BY WHICH MUST BE SATISFIED BEFORE**
117 181 THE DATA IS SELECTED FOR RETRIEVAL FROM THE DATA BASE.
118 184 CONDITIONS MAY BE SIMPLE OR COMPOUND. A COMPOUND CONDITION IS MADE UP OF
119 187 SEVERAL SIMPLE CONDITIONS LINKED BY THE CONJUNCTIONS AND/OR. THE SYSTEM
120 190 WILL PROMPT YOU TO BUILD UP COMPOUND CONDITIONS IN TERMS OF SIMPLE
121 193 CONDITIONS.
122 196 EACH SIMPLE CONDITION CONSISTS OF THREE PARTS;
123 199 THE OPERANDS ARE SEPARATED BY AN OPERATOR.
124 202 THE OPERAND IS THE NAME OF THE DOMAIN TO WHICH THE ITEMS TO BE
125 205 TESTED BELONG.
126 208 THE SECOND OPERAND SPECIFIES WHAT THE FIRST OPERAND MUST BE
UNIT: 0  
GUAC132-DAT1  
SUBIVYST  
INPUT  
DIAG:  
STAGE6 - DATABASE INCONSISTENCIES  
YOUR PROBLEM TAKES USE OF MORE THAN ONE RELATION. IF, WHEN THE RELATIONS ARE JOINED, A DATA INCONSISTENCY IN A GONIAX USED AS A SEQUENCE KEY IS DETECTED BY THE SYSTEM SUCH THAT THERE IS NOT A MATCHING KEY ITEM FOR ALL RELATIONS, WHICH ONE OF THE FOLLOWING ACTIONS DO YOU WISH THE SYSTEM TO TAKE?  
1. IGNORE THE UNMATCHED ITEMS AND CONTINUE WITH THE NEXT DATA RETRIEVAL  
2. CREATE A MESSAGE OF ERROR TOGETHER THE NEXT DATA RETRIEVAL  
3. TERMINATE THE PROCESSING OF THE DATABASE.  

PLEASE TYPE 1, 2 OR 3. THE DEFAULT REPLY IS 1.  
THE SYSTEM WILL PRINT THE RELATION NAME AND THE VALUE OF THE UNMATCHED KEY TOGETHER THE MESSAGE "MIS-MATCH". ALTERNATIVELY YOU MAY SPECIFY YOUR OWN MESSAGE OF UP TO 50 CHARACTERS. PLEASE TYPE S FOR SYSTEM  

INPUT:  
DIAG:  
STAGE6 - DATABASE INCONSISTENCIES - CHECK POINT 6  
NOTE: SET CHECK-POINT NO.  
G0PT6:  
NOTE: PRINT INTRODUCTORY PARAGUOGUE FOR STAGE 6  
#LNG=13  
CALL 2000  
DELETE AND VALIDATE RESPONSE  
CALL 9060  
IF #STC.GT.1,GOTO 3  
CALL FSTR(#G0PT,1,1)  
NOTE: INVALID RESPONSE  
OBEY 21004  
NOTE: SET OPTION CODE  
IF #OPT,NE.,',GOTO 6  
#G0PT=11  
CALLCOPY(#G0PT,#G0PT)  
CALLCOPY (#STC,#STC)  
DEPOSITO DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDRESS) WRITE  
NOTE: DEPOSITO DETECTED. PLEASE TYPE YES FOR YES OR NO FOR NO, THE DEFAULTS CALLCOPY (#STC, ADDR
I. /

ON

iiii

820

AT

ml

828

% 9

% 200

LABEL ON STAGE

CHAIN0,STAGE6

DATE NAME START, NO.; ON POSITION OF INPUT BUFFER

NOTE; #LNER = ERROR INDICATOR = INITIALLY CLEARED TO ZERO

NOTE; TEMPORARILY SAVE CONTENTS OF INPUT BUFFER

SQUASH

NOTE: CHECK NO. OF CHARACTERS KEYED

NOTE; CHECK FOR UNDERLINE BLANKS

WRITE #S02

IF LV1.EQ. '2', GO TO 207

NOTE; CHECK FIRST CHARACTER FOR HYPHEN OR Z

CALL STR1(LV1,1,1)

IF #GT1, 17, ' ', GO TO 207

NOTE; CHECK LAST CHARACTER FOR HYPHEN

CALCSTR(LV1,1,1)

IF #LST1 OR #ST1, GO TO 205

NOTE; IF #LST1, GO TO 205

WRITE #R1

SET UP LOOP TO CHECK FOR ALPHABETIC, NUMERIC AND HYPHEN ONLY

NOTE: ZEROIZE COUNT OF ALPHA CHARACTERS

ALPH=0

NOTE; EXTRACT NEXT CHARACTER

CALL STR1(LV1,1,1)

NOTE; TEST NUMERIC

IF #DIG1(LV1)

NOTE; TEST ALPHABETIC

IF #LET1(LV1)

NOTE; TEST LETTER

DEPOSIT #LV1%IN%VALID%CHARACTER,%ONLY%A%TO%Z%AND%-%ALLOWS

CALL COPY(D4,D)

WRITE

NOTE; SET ERROR INDICATOR ON

#LNER=1

NOTE; UPDATE POINTER AND TEST FOR END OF LOOP

IP #GTR1GT,E4, GOTO 214

GOTO 203

NOTE; NO ALPHABETIC CHARACTER PRESENT

DEPOSIT AT LEAST ONE ALPHA CHARACTER

NOTE; NAME TOO LONG

DEPOSIT #NAME=1

END

WRITE

#LNER=1

GOTO 206

WRITE

#LNER=1

GOTO 209

NOTE; NAME TOO LONG

DEPOSIT #NAME=1

END

WRITE

#LNER=1

GOTO 209

WRITE

#LNER=1
WRITE your #LWORD simple condition.

NOTE: CONDITIOAN FOR REASON SHOWN ABOVES.

CALL 7002

IF #LERR.EQ.1, GOTO 7006

NOTE: CHECK FOR MORE CONDITIONS TO COME.

DEPOSITO ADJ.YOUR#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWORD#LWOOD"NEXT".

NOTE: REQUEST CONDITION.

DEPOSITO IS THE CONDITION WHICH PRECEDES YOUR NEXT CONDITION "AS

CALLCOPY(57, ND"&OROR")

WRITE DEPOSITO PLEASE TYPE A FOR "AND" OR B FOR "OR" THE DEFAULT REPLIES

CALLCOPY(57, ND"&OROR")
OPERATOR SPECIFICS

THE COMPARISON BETWEEN THE FIRST AND SECOND OPERANDS IS MADE DURING THE CONDITION TEST.

OPERATOR MEANS ONE OF THE FOLLOWING:

- **EQ** EQUAL TO
- **LT** LESS THAN
- **LE** LESS THAN OR EQUAL TO
- **GT** GREATER THAN
- **GE** GREATER THAN OR EQUAL TO
- **NE** NOT EQUAL TO

THE FULL STEPS ARE AN INTEGRAL PART OF THE OPERATOR.

YOU MAY NEGATE ANY OF THE ABOVE OPERATORS BY INCLUDING THE WORD NOT IMMEDIATELY AFTER THE FIRST FULL STOP.

THE FOLLOWING STEPS ARE SOME EXAMPLES OF SIMPLE CONDITIONS:

**NOT GT, NOT GREATER THAN**

**GT, GREATER THAN**

**NOT EQ, NOT EQUAL TO**

**EQ, EQUAL TO**

**PART-NAMES**

**PART-NAME .EQ. "NAILS"**

**INPUT, INPUT**

**CALL COPY(57, ROI, THE DOMAIN)**

**WRITE**

**DEPOSIT**

**DEPOSIT IN YOUR RELATION(S)? S**

**CALL 2200**

**NOTE: READ AND VALIDATE REPLY**

**SQUASH**

**IF NOT GT, GOTO 2**

**CALL 2100**

**NOTE: INVALID RESPONSE**

**CONTINUE**

**LABEL**

**BEND**

**NOTE: PRINT SELECTION DIALOGUE**

**SELECT DIALOG**

**READ LLINE #501**

**DEPOSITO #5015**

**WRITE**

**SQUASH LLINE - LLINE 4**

**IF VALUE LE 44, GOTO 5**

**DEPOSITO DATA RETRIEVAL S**
GOTO 7163
NOTE: VALIDATE OPERATOR
RESET #S05
IF NOT AT #LOP, GOTO 7104
GOTO 7105
NOTE: NEGATED CONDITION
37104 DEPOSIT NOT AS
RESET #S01
DEPOSIT #LOPS
BEGIN #S01
IF #G0UP, GE, 2, GOTO 7140
NOTE: IDENTIFY CONDITION
37105 IF #LOP, EQ, GT, GOTO 7106
IF #LOP, EQ, LT, GOTO 7107
IF #LOP, LE, G0T0, 7110
IF #L0P, GE, LE, G0T0, 7112
IF #LOP, GE, G0T0, 7116
37140 IF DEPOSIT INVALID OPERATOR
GOTO 7104
37106 DEPOSIT GREATER THAN
GOTO 7118
37108 DEPOSIT LESS THAN
GOTO 7118
37110 DEPOSIT EQUAL
GOTO 7118
37112 IF NOT AT #S06, GOTO 713
DEPOSIT NOT EQUAL
GOTO 7118
37113 RESET #S03
37114 IF NOT AT #S08, GOTO 7115
DEPOSIT NOT GREATER THAN
GOTO 7118
37115 IF NOT AT #S08, GOTO 7115
DEPOSIT NOT LESS THAN
GOTO 7118
37117 RESET #S08
NOTE: VALIDATE SECOND OPERAND
37118 #L1P=#L1P
NOTE: SAVE LENGTH OF 2ND OPERAND
LEN
37120 #SIC
NOTE: PEG FOR ALPHANUMERIC LITERAL
37121 RESET #S04
DEPOSIT #S01
CALLSTR(#L11, 11)
37122 IF #L11 NE, "", GOTO 7122
NOTE: TEST FOR VALID LITERAL
37123 IF ALL GE 5, GOTO 7125
37124 DEPOSIT 2ND OPERAND IS %INVALID%ALPHANUMERIC LITERAL.
GOTO 7108
37125 NUM
NOTE: CHECK FOR CLOSING QUOTE
37126 CALLSTR(#L11, 11)
37127 IF #L11 EQ, "", GOTO 7130
NOTE: TEST FOR NUMERIC LITERAL
37128 NOTE: CLEAR DECIMAL PT. INDICATOR
4
27. DIAL8 AND STAGE8 SUBFILES

Input:

Stage 8 - Report Layout Introduction

Please choose your page width. A narrow page of 70 characters is
IS FOR LINE PRINTER OUTPUT. PLEASE TYPE N FOR NARROW OR M FOR WIDE.

The default reply is NARROW.

Would you like to design the format of your report or would
you prefer only to specify which items should be included in the output
and leave the system to arrange the exact format of the lines(s)?

If you design your own layout, you may include text information as well
as numeric items including the date, time and page number. You may also
specify whether or not you want to insert editing characters into data
items and improve their presentation.

The system will arrange the items for output in the required order
across whose line there is no blank space. A new line will
be included at the end of a line for the
complete item. All output items will be unedited.

Do you wish to specify the format of your report? Please type y for yes
or n for no. The default reply is no.

You will shortly be asked to design the lines which comprise the titles.

Up your data domains and temporary items will be displayed to you back at
a time. You are then required to select a single character label from the
list shown below for each position of the items you wish to print.

The labels are case insensitive. The following characters are valid labels:

A Z 0 9 . 

You will have noticed that not all alphabetic characters are available
for use as data labels. This is because some of them have special
meanings associated with editing data items prior to printing. The
NOTE: CLEAR POINT INDICATOR

NOTE: SET DECIMAL INDICATOR

NOTE: CHECK LENGTH

NOTE: EXTRACT NEXT CHARACTER

NOTE: TEST FOR DIGIT

NOTE: CHECK FOR PREVIOUS POINT

NOTE: CHECK FOR END OF LITERAL

NOTE: CHECK FOR END OF LITERAL

NOTE: TEST COMPATIBILITY OF OPERAND TYPES

NOTE: TEST COMPATIBILITY OF OPERAND TYPES

NOTE: QUALIFY NAME OF 2ND OPERAND

NOTE: GENERATE COMPL. STATEMENT

NOTE: PRINT INVALID RESPONSE MESSAGE

NOTE: PRINT PLEASE TYPE MESSAGE

NOTE: ROUTINE TO SEARCH DOMAIN INDEX FOR THE DOMAIN NAME IN #S09

NOTE: #LIND = INDEX FOUND 1=NOT FOUND
IF JGTY. NE, 1, GOTO 10
CALL FSTR(#6#N#H#L#H, 1, 2)
CALL CONV(#S#6#N#H#L#H, %H#N#H#L#H)
IF #6#N#H#L#H.LT. 4#N, GOTO 6
GOTO 6
NOTE: INVALID RESPONSE
OBEY 2100, 0
NOTE: SET DEFAULT SIZE
GOTO 6
NOTE: OFFER FORMAT OPTION
$LINE=6
RESP=6
CALL 5000
NOTE: READ AND VALIDATE REPLY
CALL 0000
SQUASH
IF #S#T#C#G#T, 1, GOTO 13
CALL FSTR(13, 1, 11)
IF #V#T#N#T, 1, GOTO 11
IF #V#L#E#Q#T, 1, GOTO 12
NOTE: INVALID RESPONSE
OBEY 2100, 0
GOTO 10
NOTE: OPTION NOT YET AVAILABLE
DEPOSIT THIS OPTION YET NOT AVAILABLE, PLEASE DESIGN YOUR OWN
CALL COPY (#S#T#F#O#T#Y#R#A#T#S)
WRITE
NOTE: PRINT DIALOGUE ABOUT DOMAIN LABELS
$LINE=62
PLSTOP=57
CALL 5000
NOTE: INITIALISE DOMAIN INDEX POINTER AND CONTROL STRINGS
RESET #503
RESET #504
DEPOSIT #A#S#F#H#J#I#K#L#N#O#P#T#U#V#W#X#Y#Z
DEPOSIT #5035
WD#P#T#1
SELECT DOMAIN INDEX
READ #D#P#T#1
NOTE: EXTRACT NAME AND TYPE
VSTR #S#0#1, 1
#G#N#P#G#C#C#7#1
CALL USER (#L#V#2, 0, 1)
#G#N#P#G#C#C#7#1
CALL USER (#L#V#5, 0, 1)
#G#N#P#G#C#C#7#1
CALL USER (#L#V#5, 0, 1)
NOTE: DISPLAY NAME AND READ LABEL CHARACTER ASSIGNED BY USER
DEPOSIT #S#0#25
WRITE
CALL 9000
NOTE: VALIDATE LABEL
SQUASH
IF #S#T#C#G#T, 1, GOTO 16
CALL FSTR(#L#V#1, 1, 12)
IF #L#V#1 L#O#T, 1, GOTO 18
IF #L#V#1 IN. #S#0#4, GOTO 17
NOTE: INVALID RESPONSE
OBEY 2100, 15
NOTE: DELETE SELECTED CHARACTER FROM #S#0#4
CALL #3#P#S#0#4, #D#P#T#1
WHILE
DEPOSIT #S#0#4
VSTR #S#0#3, 1, #L#V#1
#L#P#E#G#C#C#7#1
NOTE: SELECT LABEL TABLE AND WRITE RECORD
SELECT LABEL TABLE
SELECT #S#0#4
RESET #S#0#4
YOU HAVE DECIDED TO USE AS IT'S LABEL. THE DEFAULT REPLY, 'HAD BY PRESSING THE ACCEPT KEY U
O 547 IN THE DESIRED POSITION OF THE LINE. BLANK CHARACTERS BEFORE AND U
548 BETWEEN ITEMS SHOULD BE INDICATED BY TYPING THE # CHARACTER, BUT BLANK
557 POSITIONS AND PADDED OUT WITH BLANKS. IF TOO FEW PRINT POSITIONS ARE
558 ALLOCATED TO A DATA ITEM THEN CHARACTERS AT THE RIGHTHAND END WILL BE
559 EDITED IN A FEATURE TO BE DESCRIBED SHORTLY IS IN USE, THE

DECIMAL POINT IN THE OUTPUT FORMAT OF A NUMERIC ITEM IS ASSUMED TO BE
565 REQUIRED. THE U
569 INPUT

NOTE (NOT VALIDATE PAGE WIDTH OPTION J
575 % CALL "9000'

30 CALL F.strings

7. WRITE 'DEPOSITO.THIS MUST BE A VALUE IN THE RANGE 40 TO 99, THE DEFAULT IS 60). Note: Read

99 WRITE ': ' OBEY 2100, 1 "DEPOSITO.THE PREPARED DEPOSIT IS: X.XXX

84
SEPARATE FOLDER OF LISTINGS

CONTENTS

1. OUTLINE PROCESSING FOR THE COBOLGEN MACRO
2. INITIALISATION SECTION OF COBOLGEN MACRO
3. SKELETON COBOL PROGRAM - GOPART2 SUBFILE
4. * COMMENT PROCESSING
5. *FILE PROCESSING
6. *INL PROCESSING
7. *TITLE PROCESSING
8. *HEAD PROCESSING
9. *OUT PROCESSING
10. *GO PROCESSING
11. MESSAGES AND CREATE SUBFILES
12. PASSCHANGE SUBFILE
13. LIBRARIAN SUBFILE
14. LIBPRELIM SUBFILE
15. LIBDELETE SUBFILE
16. LIBADD SUBFILE
17. DIAL1 AND STAGE1 SUBFILES
18. DIAL2 AND STAGE2 SUBFILES
19. DIAL3 AND STAGE3 SUBFILES
20. STAGE3-1 SUBFILE
21. DIAL4 AND STAGE4 SUBFILES
22. DIAL5 AND STAGE5 SUBFILES
23. STAGE5-1 SUBFILE
24. STAGE5-2 SUBFILE
25. DIAL6 AND STAGE6 SUBFILES
26. DIAL7 AND STAGE7 SUBFILES
27. DIAL8 AND STAGE8 SUBFILES
28. DIALHELP AND HELP SUBFILES
Notes:

1. The type face of the teletype on which items 1 to 21 and 28 were produced causes £ to appear as $ and £ to appear as \

2. The parameter marker used during the Filetab to COBOL generation study was / (Items 1 to 10). The parameter marker used during the development of macros for the COBOL report program generating system was @ (Items 11 to 28).

3. The development of items 20 to 28 is incomplete.
1. OUTLINE PROCESSING FOR THE COBOLGEN MACRO

MACRO%.
BEGIN
DEF COBOLGEN(/)
NOTE: INITIALIZATION

NOTE: READ FIRST RECORD
READ
NOTE: IDENTIFY RECORD TYPE. ONLY THE
NOTE: FOLLOWING ARE VALID -
NOTE: #LVI IS USED TO CONTAIN FIRST 8 CHARACTERS
NOTE: OF INPUT BUFFER
NOTE: LITERALS ARE THEN MATCHED AGAINST #LVI
CALLFSTR(#LVI,1,8)
IF '*'.AT.#LVI,GOTO 2
IF '*FILE'.AT.#LVI,GOTO 3
IF '*INL'.AT.#LVI,GOTO 4
IF '*TITLE'.AT.#LVI,GOTO 5
IF '*HEAD'.AT.#LVI,GOTO 6
IF '*OUT'.AT.#LVI,GOTO 7
IF '*GO'.AT.#LVI,GOTO 8
NOTE: ERROR CONDITION - WRITE MESSAGE AND TERMINATE RUN
CALLCOPY(1,COBOLGEN%ERROR%CONDI)
CALLCOPY(21,TION%)
CALLCOPY(27,#LVI)
WRITE
GOTO 9999

9999 CONTINUE
END
NOTE:  INITIALISATION

NOTE:  GENERATE VARIABLE CONTAINING QUOTE

READ

CALLFSTR(#GOHO,1,1)

NOTE:  GENERATE VARIABLE CONTAINING #

#GHATC='#

NOTE:  #GV2 = DEFAULT VALUE FOR PROGRAM ID

#GV2='TABC'

NOTE:  INITIALISE MACROS ABOUT TO BE GENERATED

#GS01=0

#GFIL='NOTEPARA'
OBEY 101,11

#GFIL='TITLES'
OBEY 101,12

#GFIL='NOTEPARA'
OBEY 101,13

#GFIL='OTHERPAR'
OBEY 101,14

#GFIL='HEADINGS'
OBEY 101,15

#GFIL='INPTRECS'
OBEY 101,16

#GFIL='CTRLBRECS'
OBEY 101,17

#GFIL='SAVETOTL'
OBEY 101,18

#GFIL='ADDPARAS'

#GS15=0
OBEY 101,26

#GFIL='WRITEPARA'
OBEY 101,19

#GFIL='MOVEPARA'
OBEY 101,20

#GFIL='OUTRECS'
OBEY 101,21

#GFIL='BREAKPARA'
OBEY 101,24

#GFIL='PAGEPARA'
OBEY 101,22

#GFIL='FINALPARA'
#GS12=0
OBEY 101,1
**3. SKELETON COBOL PROGRAM - GOPART2 SUBFILE**

%MACRO%.
%BEGIN
%FILE,%GOPART2
%DEF GOPART2
% #LV1='01'
% [PART1]**** IDENTIFICATION DIVISION.
PROGRAM-ID. #GV2#LV1.
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. ICL-1907.
OBJECT-COMPUTER. ICL-1907 MEMORY SIZE 19000 WORDS.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT #S06 ASSIGN TO #GV1#GV12 1.
   SELECT LINE-PRINTER ASSIGN TO PRINTER 1.
DATA DIVISION.
FILE SECTION.
FD #S06
   BLOCK CONTAINS 2048 CHARACTERS
   LABEL RECORDS #S07
   DATA RECORD IS INREC.
01 INREC.
   02 IN-REC.
   03 FILLER PICTURE X(124) OCCURS 17.
   03 FILLER PICTURE X(8).
   [INPUTREC]
FD LINE-PRINTER
   LABEL RECORDS OMITTED
   DATA RECORD IS OUTREC.
01 OUTREC PICTURE X(124).
   [OUTREC]
WORKING-STORAGE SECTION.
77 LINE-COUNT PICTURE 99 VALUE ZERO.
77 PAGE-SWITCH PICTURE 9 VALUE ZERO.
01 CONTROL-BREAK.
   02 FILLER PICTURE X.
   [CTRLBRKE]
01 CONTROL-TOTALS.
   02 FILLER PICTURE X.
% RESET #S01
% DEPOSIT #GOUT
% READ #S01
PROCEDURE DIVISION.

IF #GSL.EQ.0,GOTO 912

IF #GSL.EQ.'L',GOTO 901

IF #GSL.EQ.'F',GOTO 904

MOVE #LV1 IN INREC TO #LV1 IN CONTROL-BREAK.

CALLFSTR(#LV1,0,1)

IF #LV1.EQ.'L',GOTO 904

IF #LV1.EQ.'F',GOTO 904

Move #LV1 in INREC to #LV1 in CONTROL-BREAK.

CALLFSTR(#LV1,0,1)

IF #LV1.EQ.'L',GOTO 904

IF #LV1.EQ.'F',GOTO 904

Move #LV1 in INREC to #LV1 in CONTROL-BREAK.

CALLFSTR(#LV1,0,1)

IF #LV1.EQ.'L',GOTO 904

IF #LV1.EQ.'F',GOTO 904

Move #LV1 in INREC to #LV1 in CONTROL-BREAK.

CALLFSTR(#LV1,0,1)

IF #LV1.EQ.'L',GOTO 904

IF #LV1.EQ.'F',GOTO 904

Move #LV1 in INREC to #LV1 in CONTROL-BREAK.

CALLFSTR(#LV1,0,1)

IF #LV1.EQ.'L',GOTO 904

IF #LV1.EQ.'F',GOTO 904

Move #LV1 in INREC to #LV1 in CONTROL-BREAK.

CALLFSTR(#LV1,0,1)

IF #LV1.EQ.'L',GOTO 904

IF #LV1.EQ.'F',GOTO 904

Move #LV1 in INREC to #LV1 in CONTROL-BREAK.
PERFORM PARA-ADD.

2914 CONTINUE
  GO TO PARA-2.
  PARA-3.
  PERFORM PARA-BREAK THRU PARA-BREAK-EXIT.
  IF #G012.EQ.0, GOTO 913
  PERFORM PARA-FINAL.

2913 CONTINUE
  CLOSE #G06
  LINE-PRINTER.
  STOP RUN.

  IF #G015.EQ.0, GOTO 907
  PARA-ADD.

2907 CONTINUE
  IF #G003.EQ.0, GOTO 908
  PARA-TITLE.

2908 CONTINUE
  PARA-PAGE.
  PARA-MOVE.
  PARA-WRITE.
  PARA-BREAK.
  IF #G012.EQ.0, GOTO 909
  PARA-FINAL.

2909 CONTINUE
  IF ' L', IN .#G06D, GOTO 910
  PARA-LINE-CHECK. EXIT.

2910 CONTINUE
  PARA-LINE-CHECK. IF LINE-COUNT GREATER THAN #G0Z PERFORM
  PARA-PAGE, MOVE 1 TO PAGE-SWITCH, GO TO PARA-LINE-EXIT.

  MOVE 0 TO PAGE-SWITCH.

2911 CONTINUE
  PARA-LINE-EXIT. EXIT.

[OTHERPAR]

****

2999 CONTINUE

END
4. * COMMENT PROCESSING

% NOTE: PROCESS * COMMENT RECORDS
% NOTE: PREPARE TO WRITE ON NOTEPARA SUBFILE
% #GFIL='NOTEPARA'
% OBEY 102,201
% NOTE: TEST FOR FIRST COMMENT
%201 IF #GS01.NE.0,GOTO 202
% NOTE: WRITE PARAGRAPH STATEMENT ON NOTEPARA SUBFILE
% NOTE-PARA. NOTE
% NOTE: SET #GS01 TO BE NON-ZERO
% #GS01=1
% NOTE: SAVE COMMENT CHARACTERS 3-6 IN #GV2 AS PROGRAM ID
% CALLFSTR(#GV2,3,4)
% NOTE: EXTRACT FIRST 60 CHARACTERS
%202 CALLFSTR(#LV1,1,8)
% CALLFSTR(#LV2,9,8)
% CALLFSTR(#LV3,17,8)
% CALLFSTR(#LV4,25,8)
% CALLFSTR(#LV5,33,8)
% CALLFSTR(#LV6,41,8)
% CALLFSTR(#LV7,49,8)
% CALLFSTR(#LV8,57,4)
% NOTE: OUTPUT COMMENT TO NOTEPARA SUBFILE
% #LV1#LV2#LV3#LV4#LV5#LV6#LV7#LV8
% NOTE: EXTRACT CHARACTERS 61-72
% CALLFSTR(#LV1,61,8)
% CALLFSTR(#LV2,69,4)
% #LV1#LV2
% GOTO 1
NOTE: ROUTINE FOR PROCESSING *FILE DIRECTIVES

NOTE: #S06 USED FOR FILE NAME

NOTE: #S07 = LABEL OPTION

NOTE: RESET #S01 FOR USE AS TEMPORARY STORE

RESET #S06

RESET #S01

NOTE: START GENERATING CONSTANTS MACRO

/1/FILE,0,CONSTANTS/
/1/DEF CONSTANTS
/1/  RESET #GHATCS06
/1/  RESET #GHATCS07

NOTE: TEST PERIPHERAL OPTION FOR CARDS OR TAPE

CALLFSTR(#LV1,7,2)

IF #LV1.EQ.'CR',GOTO 301

IF #LV1.EQ.'MT',GOTO 302

NOTE: ERROR CONDITION ENCOUNTERED, WRITE ERROR MESSAGE

CALLCOPY(1,TAPEFILEERROR)

WRITE

GOTO 306

NOTE: #GV11 AND #GV12 USED FOR INPUT MEDIUM SPECIFICATION

#GV11='CARD-REA'

#GV12='DER'

NOTE: TEST POSITION 10 FOR PRESENCE OF FILENAME

CALLFSTR(#LV1,10,1)

IF #LV1.EQ. ' ',GOTO 303

GOTO 344

NOTE: SET #S06 = DEFAULT FILE NAME

NOTE: SET #S07 = LABEL OPTION

DEPOSIT6 DEFAULT-NAME/
/1/  DEPOSIT7 OMITTED/

GOTO 306

301 #GV11='CARD-REA'

#GV12='DER'

NOTE: TEST POSITION 10 FOR PRESENCE OF FILENAME

CALLFSTR(#LV1,10,1)

IF #LV1.EQ. ' ',GOTO 303

GOTO 344

NOTE: SET #S06 = DEFAULT FILE NAME

NOTE: SET #S07 = LABEL OPTION

DEPOSIT6 DEFAULT-NAME/
/1/  DEPOSIT7 OMITTED/

GOTO 306

NOTE: EXTRACT AND STORE FILE NAME

#GINP=10

307 CALLFSTR(#LV1,0,1)

IF #LV1.EQ. ' ',GOTO 308

DEPOSIT6 #LV1/

DEPOSIT6 #LV1/

#GINP=#GINP+1

IF #GINP.LT.22,GOTO 307

IF #GINP.EQ.11,GOTO 309

READ #S01

SQUASH

IF #GSTC.EQ.0,GOTO 309

/1/  DEPOSIT7 STANDARD/1/VALUE/1/OF/1/ID/1/"#S06"

GOTO 346

CONTINUE

/1/  DEPOSIT7 STANDARD/1/VALUE/1/OF/1/ID/1/

/1/  DEPOSIT7 "/1/1/1/1/1/1/1/1/1/1/1/"

RE SET #S06

DEPOSIT6 DEFAULT-NAME/

CONTINUE

/1/  DEPOSIT6 #S06/

/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1/

GOTO 1
6. *INL PROCESSING

```
% NOTE: ROUTINE FOR PROCESSING *INL DIRECTIVES
% NOTE: AND PARAMETERS
% NOTE: SET FIELD COUNT TO 0
% #GFCT=0
% READ
% NOTE: READ RECORD AND SET #GINP=0
% #GINP=0
% NOTE: TEST FOR * RECORD
% CALLFSTR(#LV1,1,1)
% IF #LV1.EQ. '*', GOTO 10
% NOTE: SQUASH INPUT BUFFER AND SAVE LENGTH IN #LSTC
% SQUASH
% #LSTC=#GSTC
% NOTE: LOCATE AND STORE FIELD SPECIFIER IN #GVS
% #GINP=#GINP+1
% CALLFSTR(#GVS,1,1)
% NOTE: ADD 1 TO FIELD COUNT
% #GFCT=#GFCT+1
% NOTE: RESET #S01 AND STORE STRING OF NUMERIC START POSITION
% NOTE: CHARACTERS IN IT
% RESET #S01
% NOTE: INCREASE #GINP, LOOK AT NEXT CHARACTER AND TEST FOR NUMERIC
% #GINP=#GINP+1
% CALLFSTR(#LV1,1,1)
% TESTDIGIT(#LV1)
% IF #GSTS.NE.0, GOTO 403
% DEPOSIT #LV1
% GOTO 402
% NOTE: TRANSFER CONTENTS OF #S01 TO #GVS USING #S09 AS IN 
% NOTE: INTERMEDIATE STORAGE. SAVE #GINP IN #LINP
% #GINP=#LINP
% WRITE #S09
% #LINP=#GINP
% READ #S01
% SQUASH
% #Goup=#GSTC
% CALLFSTR(#GVS,1,1)
% READ #S09
% SQUASH
```
% #GINP=#LINP
% NOTE: STORE TYPE CHARACTER IN #GU6
% #GU6=#LV1
% NOTE: RESET #S01 AND STORE LENGTH OR END SPECIFIER IN IT
% RESET #S01
% NOTE: INCREASE #GINP, LOOK AT NEXT CHARACTER AND TEST FOR NUMERIC
% #GINP=#GINP+1
% CALLFSTR(#LV1,3,1)
% TESTDIG(#LV1)
% IF #GTEST.NE.0,GOTO 404
% DEPOSIT #LV1\%NOTE: CHECK FOR END OF BUFFER
% IF #GINP.GE.#LSTC,GOTO 404
% GOTO 405
% NOTE: TRANSFER CONTENTS OF #S01 TO #GU7 USING #S09, SAVE #GINP IN
% NOTE: #LINP
% #GINP=#LINP+1
% CALLFSTR(#LV1,3,1)
% TESTDIG(#LV1)
% IF #GTEST.NE.0,GOTO 404
% DEPOSIT #LV1\%NOTE: CHECK FOR END OF BUFFER
% IF #GINP.GE.#LSTC,GOTO 404
% GOTO 405
% NOTE: TRANSFER CONTENTS OF #S01 TO #GU7 USING #S09, SAVE #GINP IN
% NOTE: #LINP
% #GINP=#LINP+1
% CALLFSTR(#LV1,3,1)
% TESTDIG(#LV1)
% IF #GTEST.NE.0,GOTO 404
% DEPOSIT #LV1\%NOTE: CHECK FOR END OF BUFFER
% IF #GINP.GE.#LSTC,GOTO 404
% GOTO 405
% NOTE: TRANSFER CONTENTS OF #S01 TO #GU7 USING #S09, SAVE #GINP IN
% NOTE: #LINP
% #GINP=#LINP+1
% CALLFSTR(#LV1,3,1)
% TESTDIG(#LV1)
% IF #GTEST.NE.0,GOTO 404
% DEPOSIT #LV1\%NOTE: CHECK FOR END OF BUFFER
% IF #GINP.GE.#LSTC,GOTO 404
% GOTO 405
% NOTE: TRANSFER CONTENTS OF #S01 TO #GU7 USING #S09, SAVE #GINP IN
% NOTE: #LINP
% #GINP=#LINP+1
% CALLFSTR(#LV1,3,1)
% TESTDIG(#LV1)
% IF #GTEST.NE.0,GOTO 404
% DEPOSIT #LV1\%NOTE: CHECK FOR END OF BUFFER
% IF #GINP.GE.#LSTC,GOTO 404
% GOTO 405
% NOTE: TRANSFER CONTENTS OF #S01 TO #GU7 USING #S09, SAVE #GINP IN
% NOTE: #LINP
%423 IF #GUS.EQ.4, GOTO 429
% NOTE: GENERATE INITIAL FILLER STATEMENT(S)
% #LV1=407
% GOTO 433
%409 CONTINUE
/1/1/1/1/1/1/1/1/1/
% NOTE: TEST FIELD SPECIFIER FOR TOTALLING FIELD
%407 TESTDIGT(#GUA)
% IF #GTST.NE.3, GOTO 428
% NOTE: WRITE CONTROL ITEM ON SAUETOTL SUBFILE
% #GFIL='SAUETOTL'
% OBEY 102, 425
%425 CONTINUE
%3 A#GUA PICTURE 9(15) COMPUTATIONAL VALUE ZERO.
/1/1/1/1/1/1/1/1/1/1/
% NOTE: WRITE A3 LEVEL STATEMENT ON INPUTREC SUBFILE
% #GFIL='INPUTREC'
% OBEY 102, 426
%426 CONTINUE
%3 A#GUA PICTURE 9(#GUA),
/1/1/1/1/1/1/1/1/1/1/
% NOTE: WRITE ADD PARAGRAPH STATEMENTS
% #GFIL='ADDPARA'
% #GS15=1
% OBEY 142, 444
%444 CONTINUE
ADD A#GUA IN INREC TO A#GUA IN 1-LEVEL,
/1/1/1/1/1/1/1/1/1/1/
% GOTO 414
% NOTE: TEST FOR CONTROL BREAK ITEM
%448 IF #GUA.IN., 'MNO^OR', GOTO 414
% GOTO 413
% NOTE: WRITE CONTROL BREAK ITEM ON CTRLBKKE SUBFILE
% #GFIL='CTRLBKKE'
% OBEY 142, 427
%427 CONTINUE
%2 #GUA PICTURE X(#GUA),
/1/1/1/1/1/1/1/1/1/1/
% NOTE: WRITE A3 LEVEL STATEMENT ON INPUTREC SUBFILE
% #GFIL='INPUTREC'
% OBEY 102, 428
%428 CONTINUE
%3 #GUA PICTURE X(#GUA),
/1/1/1/1/1/1/1/1/1/1/
% NOTE: GENERATE FINAL FILLER STATEMENT
% 2414  #LV1=244H-#GUS-#GUS
% 2415  IF #LV1.EQ.0,GOTO 415
% 2416  #GFILE='INPUTREC'
% 2417  OKEY 192,429
% 2429  #LV3=415
% 2433  NOTE: GENERATE FILLER STATEMENT(S)
% 2434  #LV2=0
% 2444  IF #LV1.LT.124,GOTO 431
% 2445  #LV1=#LV1-120
% 2446  #LV2=#LV2+1
% 2447  GOTO 424
% 2451  IF #LV1.EQ.0,GOTO 452
% 2452  43 FILLER PICTURE X(#LV1).
% 2453  IF #LV2.EQ.4,GOTO 434
% 2454  IF #LV2.EQ.1,GOTO 453
% 2455  43 FILLER PICTURE X(124) OCCURS #LV2.
% 2456  GOTO 434
% 2459  CONTINUE
% 2460  43 FILLER PICTURE X(124).
% 2463  CONTINUE:
% 2464  /1//1//1//1//1//1//1//1/1/
% 2467  GOTO #LV3
% 2473  NOTE: TEST FOR END OF INPUT BUFFER
% 2475  IF #GINP.GE.#LSTG,GOTO 401
% 2478  GOTO 411
7. *TITLE PROCESSING

NOTE: ROUTINE FOR PROCESSING *TITLE DIRECTIVES AND PARAMETERS

NOTE: SET TITLEPAR SWITCH #GS03 = 1

#GS03 = 1

NOTE: SQUASH INPUT BUFFER AND EXTRACT SKIP BEFORE AND SKIP AFTER

NOTE: COUNTS AFTER SETTING DEFAULT VALUES USING #GV4 AND #GV5

#GV4 = '1'

#GV5 = '3'

CALL VSTRC AGV/4, 7, ,)

AGINP = AGSTC + 7

IF #GSTC .EQ. #INP, GOTO 591

#INP = #INP + 1

CALL VSTRC (AGV5, A, ,)

NOTE: SET TITLE LINE COUNTER #GV6 = 1

#GV6 = 1

NOTE: REMOVE UNWANTED SPACES FROM #GV4 AND #GV5 USING #SA9

RESET #SA9

DEPOSIT #GV4

READ #SA9

SQUASH

#GOUP = #GSTC

CALLFSTR (#GV4, 1, I)

RESET #SA9

DEPOSIT #GV5

READ #SA9

SQUASH

#GOUP = #GSTC

NOTE: READ PARAMETER RECORD AND TEST FOR *RECORD USING #LV1

#502 READ

CALLFSTR (#LV1, 1, I)

IF #LV1 .EQ. '*, GOTO 593

NOTE: GENERATE #1 LEVEL ENTRY

#506 #GFILE = 'TITLES'

OKAY 102, 510

#510 CONTINUE
NOTE: EXTRACT FIRST 58 CHARACTERS FROM INPUT BUFFER
CALLFSTR(#LV1,1,M)
CALLFSTR(#LV2,9,8)
CALLFSTR(#LV3,17,8)
CALLFSTR(#LV4,25,8)
CALLFSTR(#LV5,33,8)
CALLFSTR(#LV6,41,8)
CALLFSTR(#LV7,49,8)
CALLFSTR(#LV8,57,2)
NOTE: GENERATE 42 LEVEL ENTRY
42 FILLER PICTURE X(58) VALUE "#LV1#LV2#LV3#LV4#LV5#LV6#LV7#LV8".
NOTE: EXTRACT CHARACTERS 59-72
NOTE: FROM INPUT BUFFER
CALLFSTR(#LV1,59,8)
CALLFSTR(#LV2,67,8)
NOTE: GENERATE 42 LEVEL ENTRY
42 FILLER PICTURE X(14) VALUE "#LV1#LV2".
NOTE: READ RECORD AND TEST FOR * IN POSITION 1
READ
CALLFSTR(#LV1,1,1)
IF #LV1.EQ.1,*','GOTO 514
CALLFSTR(#LV1,1,1)
CALLFSTR(#LV2,9,8)
CALLFSTR(#LV3,17,8)
CALLFSTR(#LV4,25,8)
CALLFSTR(#LV5,33,8)
CALLFSTR(#LV6,41,8)
NOTE: GENERATE 42 LEVEL ENTRY
42 FILLER PICTURE X(14) VALUE "#LV1#LV2#LV3#LV4#LV5#LV6".
NOTE: READ RECORD AND EXTRACT 1ST CHARACTER IN #LV1
READ
CALLFSTR(#LV1,1,1)
NOTE: END OF LEVEL ENTRY GENERATION
CONTINUE
//1//1/1/1/1/1/1/1/1/1/
#GFILE='TITLEPAR'
OBEY 102,512
NOTE: TEST FOR FIRST LINE
512 IF #GFILE.EQ.1,'GOTO 547
NOTE: GENERATE WRITE STATEMENT FOR TITLE LINE NOT THE FIRST WRITE OUTREC FROM TITLE-#GUA AFTER ADVANCING 1 AND 1 TO LINE-COUNT.
IF #LUI.EQ.*",GOTO 503

NOTE: NOT A * RECORD - FINISH TITLEPAR GENERATION

#GU6=#GU6+1

GOTO 504

NOTE: GENERATE FILLER OF SPACES TO MAKE UP 120 CHARACTER LINE

CONTINUE

$2 FILLER PICTURE X(48) VALUE SPACES.

GOTO 505

NOTE: GENERATE WRITE STATEMENT FOR FIRST TITLE LINE

CONTINUE

WRITE OUTREC FROM TITLE-1 AFTER ADVANCING #GU4.
ADD #GU4 TO LINE-COUNT.

GOTO 506

NOTE: GENERATE STATEMENTS FOR SPACING AFTER TITLE

CONTINUE

IF #GVS.EQ.*",GOTO 507

MOVE SPACES TO OUTREC.
WRITE OUTREC AFTER ADVANCING #GVS.
ADD #GVS TO LINE-COUNT.

CONTINUE

GOTO 508

CONTINUE

GOTO 14
8. *HEAD PROCESSING

% NOTE: ROUTINE FOR PROCESSING *HEAD DIRECTIVE AND PARAMETER RECORDS
% NOTE: INITIALISE #GHED TO SPACES
% #GHED=#SPC
% NOTE: SET DEFAULT VALUES FOR VARIABLES
% NOTE: #GV4 = CONTROL LEVEL
% NOTE: #GV5 = SKIP BEFORE COUNT
% NOTE: #GV6 = SKIP AFTER COUNT
% NOTE: #GVZ = PAGE LIMIT
% #GV4='L'
% #GV5='1'
% #GV6='1'
% #GVZ='42'
% NOTE: SQUASH INPUT BUFFER
SQUASH
% NOTE: TEST FOR END OF BUFFER
IF #GSTC.EQ.5,GOTO 602
% NOTE: COPY CONTROL LEVEL INTO #GV4
CALFSTR(#GV4,6,1)
% NOTE: TEST FOR END OF BUFFER
IF #GSTC.EQ.6,GOTO 602
% NOTE: COPY SKIP BEFORE COUNT INTO #GV5
CALLUSTR(#GV5,7,)
% NOTE: TEST FOR END OF BUFFER
#Ginp=#GCCT+7
IF #Ginp.GE.#GSTC,GOTO 602
% NOTE: COPY SKIP AFTER COUNT INTO #GV6
#Ginp=#Ginp+1
CALLUSTR(#GV6,9,)
% NOTE: IF LEVEL 'L' TEST FOR NO. OF LINES PARAMETER
IF #GV4.NE.'L',GOTO 602
#Ginp=#Ginp+GCCT
IF #Ginp.GE.#GSTC,GOTO 602
% NOTE: COPY NO. OF LINES PER PAGE INTO #GVZ
#Ginp=#Ginp+1
CALLUSTR(#GVZ,0,)
% NOTE: SET HEADING LINE COUNT #GHCT = 1
#GHCT=1
% NOTE: SAVE HEADING LEVEL #GV4 IN #GHED USING #S09
RESET #S09
DEPOSIT #GV4#GHED\nREAD #S09
CALLFSTR(#GHED,1,8)
NOTE: REMOVE UNWANTED SPACES FROM #GVS, #GV5, AND #GV7 USING #S39
RESET #S49
DEPOSIT9 #GVS
READ #S49
SQUASH
#GOUP=#GSTC
CALLFSTR(#GVS,1,0)
RESET #S49
DEPOSIT9 #GV4\nREAD #S49
SQUASH
#GOUP=#GSTC
CALLFSTR(#GV4,1,4)
RESET #S49
DEPOSIT9 #GV5\nREAD #S49
SQUASH
#GOUP=#GSTC
CALLFSTR(#GV5,1,4)
NOTE: GENERATE PARAGRAPH HEADING AND LEVEL 31 ENTRY
#GFL= 'OTHERPAR'
OBEY 142,649
CONTINUE
#GV4-HEAD-WRITE
/1/1/1/1/1/1/1/1/1/1/1/
NOTE: READ RECORD
READ
#GFL= 'HEADINGS'
OBEY 142,611
CONTINUE
#GFL= 'HEAD-CHARCT'
NOTE: EXTRACT FIRST 58 CHARACTERS
CALLFSTR(#LV1,1,8)
CALLFSTR(#LV2,9,8)
CALLFSTR(#LV3,17,4)
CALLFSTR(#LV4,25,8)
CALLFSTR(#LV5,33,8)
CALLFSTR(#LV6,41,4)
CALLFSTR(#LV7,49,8)
CALLFSTR(#LV8,57,2)
FILLER PICTURE X(58) VALUE "#LV1#LV2#LV3#LV4#LV5#LV6#LV7#LV8".
NOTE: EXTRACT CHARACTERS 59 - 72
CALLFSTR(#LV1,59,8)
CALLFSTR(#LV2,67,6)
FILLER PICTURE X(14) VALUE "#LV1#LV2".
NOTE: READ RECORD AND TEST FOR * RECORD
READ
CALLFSTR(#LV1,1,1)
IF #LV1.EQ. ' ', GOTO 604
NOTE: EXTRACT CHARACTERS 1 TO 48
CALLFSTR(#LV1,1,8)
CALLFSTR(#LV2,9,8)
CALLFSTR(#LV3,17,8)
CALLFSTR(#LV4,25,8)
CALLFSTR(#LV5,33,8)
CALLFSTR(#LV6,41,8)
02 FILLER PICTURE X(48) VALUE "#LV1#LV2#LV3#LV4#LV5#LV6".
NOTE: READ NEXT RECORD AND TEST FOR *
READ
CALLFSTR(#LV1,1,1)
IF #LV1.EQ. ' ', GOTO 605
/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1
NOTE: GENERATE WRITE Verify STATEMENTS
#LV1=624
GOTO 624
#GHT=#GHT+1
GOTO 603
NOTE: GENERATE SPACE FILLER FOR MISSING SECOND RECORD
CONTINUE
02 FILLER PICTURE X(48) VALUE SPACES.
CONTINUE
#LV3=420
GOTO 621
NOTE: TEST SPACING AFTER OPTION
6220 IF #LV6.EQ. ' ', GOTO 644
NOTE: GENERATE SPACING AFTER STATEMENTS
#GFIL='OTHERPAR'
OBEY 102, 616
CONTINUE
SPACE FILLER FOR MISSING SECOND RECORD
NOTE: TEST FOR *HEAD RECORD
638 CALLFSTR(#LV1,1,8)
IF *HEAD*.AT.#LV1, GOTO 601
GOTO 10
6221 #GFIL='OTHERPAR'
OBEY 102, 616
MOVE SPACES TO OUTREC.
WRITE OUTREC AFTER ADVANCING #LV6.
ADD #LV6 TO LINE-COUNT.
/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1
NOTE: TEST FOR *HEAD RECORD
638 CALLFSTR(#LV1,1,8)
IF *HEAD*.AT.#LV1, GOTO 601
GOTO 10
6221 #GFIL='OTHERPAR'
OBEY 102, 616
% note: test for first line of heading
% 618 IF #GHCT.EQ.1, GOTO 619
   WRITE OUTREC FROM #GU4-HEAD-#GHCT AFTER ADVANCING 1.
   ADD 1 TO LINE-COUNT.
% 619 GOTO 622
% 619 CONTINUE
   WRITE OUTREC FROM #GU4-HEAD-#GHCT AFTER ADVANCING #GVS.
   ADD #GVS TO LINE-COUNT.
% 622 CONTINUE
   /1/1/1/1/1/1/1/1/1/1/1/1/
% 623 GOTO #L113
9. *OUT PROCESSING

NOTE: ROUTINE FOR PROCESSING *OUT DIRECTIVES AND PARAMETERS

NOTE: INITIALISE #GOUT TO SPACES

#GOUT=#GSPC

NOTE: SET DEFAULT VALUES FOR VARIABLES

NOTE: #GV4=CONTROL LEVEL

NOTE: #GV5=SKIP BEFORE COUNT

NOTE: #GV6=SKIP AFTER COUNT

#GV4='L'

#GV5='1'

#GV6='1'

NOTE: SQUASH INPUT BUFFER

SQUASH

NOTE: TEST FOR END OF BUFFER

IF #GSTC.EQ.4,G0T0 702

NOTE: COPY CONTROL LEVEL INTO #GV4

CALLFSTR(#GV4,5,1)

NOTE: TEST FOR END OF BUFFER

IF #GSTC.EQ.5,G0T0 702

NOTE: COPY SKIP BEFORE COUNT INTO #GV5

CALLVSTR(#GV5,6,,)

NOTE: TEST FOR END OF BUFFER

#GINP=#GCT+5

IF #GINP.GE.#GSTC,G0T0 702

NOTE: COPY SKIP AFTER COUNT INTO #GV6

#GINP=#GINP+1

CALLVSTR(#GV6,0,,)

NOTE: SAVE OUTPUT LEVEL #GV4 IN #GOUT USING #S01

RESET #S01

DEPOSIT #GV4#GOUT

READ #S01

CALLFSTR(#G0UT,1,8)

NOTE: SET OUTPUT LINE #GOCT = 0

#GOCT=0

NOTE: SET PARAGRAPH SWITCH #GPSw=0

#GPSw=0

NOTE: REMOVE UNWANTED SPACES FROM #GV5 AND #GV6 USING #S01

RESET #S01

DEPOSIT #GV5

READ #S01
CALLFSTR(#GV5,1,0)
RESET #S01
DEPOSIT #GV6
READ #S01
SQUASH
CALLFSTR(#GV6,1,0)
NOTE: READ RECORD AND SET #GINP = 1
READ
#GINP=1
NOTE: INCREMENT LINE COUNT AND SET INDICATOR #LV2 TO 0
#GOCT=#GOCT+1
#LV2=0
NOTE: SET MOVE SWITCH #GMVE = 0
#GMVE=0
#GFIL='QUTREC'
OBEY 102,710
NOTE: GENERATE LEVEL 01 ENTRY
01 #GV4-RECP#GOCT.
CONTINUE
NOTE: EXTRACT NEXT CHARACTER FROM INPUT BUFFER IN #LV1
#LV3=703
GOTO 760
NOTE: TEST #LV1 FOR BLANK
703 IF #LV1.EQ.' ',GOTO 704
NOTE: TEST FOR START OF LITERAL
704 IF #LV1.EQ.#GO10,GOTO 705
NOTE: SAVE CHARACTER IN #GV9 AND START CHARACTER COUNT #GV7
#GV9=PLV1
#GV7=1
NOTE: EXTRACT NEXT CHARACTER
706 #LV3=707
GOTO 760
NOTE: TEST FOR END OF LINE
707 IF #LV2.GT.1,GOTO 708
NOTE: TEST FOR SAME CHARACTER AS PREVIOUS
708 IF #GV9.NE.#LV1,GOTO 708
NOTE: INCREMENT CHARACTER COUNT
#GV7=#GV7+1
GOTO 706
NOTE: GENERATE #2 LEVEL ENTRY

#GFIL='OUTREC'

NOTE: TEST #GU9 FOR 0-9

TESTDIGT(#GU9)

IF #GTEST.EQ.9,GOTO 709

#GU9 PICTURE X(#GU7).

GOTO 712

#L7= #GU7-1

IF #L7.EQ.1,GOTO 777

#GU9 PICTURE Z(#L7).9.

GOTO 712

CONTINUE

NOTE: TEST AGV9 FOR 0-9

TESTDIGT(#GV9)

IF #GTST.EQ.0,GOTO 709

#GU9 PICTURE X(#GU7).

GOTO 712

CONTINUE

NOTE: FINISH WRITING TO FILE

#L7=#GU7-1

IF #L7.EQ.1,GOTO 777

#GU9 PICTURE Z(#L7).9.

GOTO 712

CONTINUE

NOTE: GENERATE MOVE STATEMENTS AND PARAGRAPH NAME IF AT LEVEL L

NOTE: TEST FOR 'L' IN #GU4

IF #GU4.EQ.'L',GOTO 714

NOTE: TEST PARAGRAPH SWITCH #GPSW

IF #GPSW.NE.0,GOTO 766

#GFIL='OTHERPAR'

OBEY 142,714

CONTINUE

#GU4-TOTALS.

#GPSW=1

#GFIL='OTHERPAR'

OBEY 142,717

CONTINUE

MOVE CORRESPONDING #GU4-LEVEL TO #GU4-REC#G0CT.

#GU4=1

GOTO 713

#GFIL='MOVEPARA'

OBEY 142,719

TESTDIGT(#GU9)

IF #GTEST.EQ.9,GOTO 724

MOVE #GU9 IN INREC TO #GU9 IN L-REC#G0CT.

GOTO 718

CONTINUE

MOVE A#GU9 IN INREC TO A#GU9 IN L-REC#G0CT.
NOTE: TEST #LV2 FOR END OF INPUT BUFFER

CONTINUE

IF #LV2.LE.1, GOTO 703

NOTE: TEST FOR CONTROL LEVEL 1 AND SET FILE NAME IN #GFIL

IF #GV4.EQ. 'L', GOTO 721

#GFIL='OTHERPAR'

GOTO 722

#GFIL='WRITEPAR'

OBEY 102, 725

NOTE: GENERATE WRITE STATEMENTS

IF #GOCT.EQ.1, GOTO 723

PERFORM PARA-LINE-CHECK THRU PARA-LINE-EXIT.
WRITE OUTREC FROM #GV4-REC#GOCT AFTER ADVANCING 1.
ADD 1 TO LINE-COUNT.

GOTO 724

CONTINUE

PERFORM PARA-LINE-CHECK THRU PARA-LINE-EXIT.
WRITE OUTREC FROM #GV4-REC#GOCT AFTER ADVANCING #GV4.
ADD #GV4 TO LINE-COUNT.

IF #GV4.EQ. 'L', GOTO 724

PERFORM #GV4-ADD.

NOTE: TEST FOR END OF LEVEL

CONTINUE

IF #LV2.LE.2, GOTO 747

NOTE: GENERATE FINAL SPACING STATEMENT

CALL CONV (#GV4, #GV4)

IF #GV4.LE.4, GOTO 730

IF #GV4.EQ. 'L', GOTO 726

#GFIL='OTHERPAR'

GOTO 727

#GFIL='WRITEPAR'

OBEY 102, 728

CONTINUE

MOVE SPACES TO OUTREC.
WRITE OUTREC AFTER ADVANCING #GV4.
ADD #GV4 TO LINE-COUNT.

GOTO 729

CALL FSTR (#LV1,1,4)

IF #LV1.EQ. '*OUT', GOTO 701

GOTO 10

NOTE: PROCESS LITERAL FIELD

RESET #SV4
NOTE: EXTRACT NEXT CHARACTER AND TEST FOR WHITE
GOTO 743
IF #LV1.EQ..GOTO, GOTO 734
DEPOSIT #LV1
#GU7=#GU7+1
GOTO 733
NOTE: GENERATE FILLER CONTAINING LITERAL
#GFIL='OUTREC'
#GU7=#GU7+2
OBEY 192, 735
CONTINUE

42 FILLER PICTURE X(#GU7) VALUE " " 549 ".

GOTO 744
NOTE: PROCESS BLANK FIELD
#GU7=1
NOTE: EXTRACT NEXT CHARACTER
#LV3=734
GOTO 744
NOTE: TEST FOR END OF INPUT BUFFER
IF #LV2.GT.1, GOTO 737
NOTE: TEST FOR BLANK CHARACTER IN #LV1
IF #LV1.EQ..", GOTO 738
NOTE: GENERATE BLANK FILLER
#GFIL='OUTREC'
OBEY 192, 739
CONTINUE

42 FILLER PICTURE X(#GU7) VALUE SPACES.
NOTE: INCREMENT BLANK CHARACTER COUNT
#GU7=#GU7+1
GOTO 744
NOTE: ROUTINE TO EXTRACT CHARACTERS FROM OUT PARAMETER RECORDS
IF #LV2.NE.9, GOTO761
IF #GINP.GT.72, GOTO 762
NOTE: FETCH NEXT CHARACTER
CALLFSTR(#LV1,3,1)
#GINP=#GINP+1
GOTO #LV3
IF #GINP.GT.48, GOTO 765
GOTO 764
READ
#GINP=1
CALLFSTR(#LV1,1,1)
IF #LV1.EQ.."", GOTO 766
#LV2=2
GOTO 763
#LV2=3
GOTO 763
READ
#GINP=1
CALLFSTR(#LV1,1,1)
IF #LV1.EQ.."", GOTO 766
#LV2=1
GOTO 764
10. *GO PROCESSING

% NOTE: ROUTINE FOR PROCESSING *GO DIRECTIVES AND FOR
% NOTE: MORE COBOL STATEMENTS FOR THE PROCEDURE DIVISION
% NOTE: IF NOTE PARAGRAPH HAS BEEN GENERATED ADD FINAL
% NOTE: FULLSTOP
% IF #GS01.EQ.0,GOTO 801
% #GFIL='NOTE PARA'
% OBEY 102,802
% CONTINUE

/1//1/1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1///1 ///
% #GS13=1
% OBEY 102,803
%803 CONTINUE

PERFORM #LVI-HEAD THRU #LVI-HEAD-EXIT.
/1/1/1/1/1/1/1/1/1/1/1/
% #GFILE='OTHERPAR'
% OBEY 102,878
%878 CONTINUE

#LVI-HEAD.
PERFORM PARA-LINE-CHECK THRU PARA-LINE-EXIT.
IF PAGE-SWITCH NOT EQUAL TO ZERO GO TO #LVI-HEAD-EXIT.

PERFORM #LVI-HEAD-WRITE.
#LVI-HEAD-EXIT. EXIT
/1/1/1/1/1/1/1/1/1/1/1/

GOTO #LVI3

NOTE: GENERATE COBOL STATEMENTS FOR PROCESSING CONTROL TOTALS
%814 IF 'M'.IN.,GOUT,GOTO #822
%823 IF 'N'.IN.,GOUT,GOTO #824
%825 IF 'O'.IN.,GOUT,GOTO #826
%827 IF 'P'.IN.,GOUT,GOTO #828
%829 IF 'R'.IN.,GOUT,GOTO #830
%831 IF 'S'.IN.,GOUT,GOTO #832
%833 IF 'F'.IN.,GOUT,GOTO #834

GOTO #840
%822 #LVI='4'
% #LV2='N'
% #LV3=#823
% GOTO #840
%824 #LVI='N'
% #LV2='O'
% #LV3=#825
% GOTO #840
%826 #LVI='O'
% #LV2='P'
% #LV3=#827
% GOTO #840
%828 #LVI='P'
% #LV2='Q'
% #LV3=#829
% GOTO #840
%830 #LVI='Q'
% #LV2='R'
% #LV3=#831
% GOTO #840
%832 #LVI='R'
% #LV3=#833
% GOTO 840
% 834 #LV1='F'
% #LV3=860
% #GFIL='FINALPAR'
% #S12=1
% OBEY 102,835
%835 CONTINUE

PERFORM #LV1-TOTALS.
111111111111111111111111
% IF #LV1.EQ.'F', GOTO 860
% #GFIL='OTHERPAR'
% OBEY 142,837
%837 CONTINUE

#LV1-ADD.
% IF #LV1.EQ.'R', GOTO 838
% IF #LV2.IN.,#GOUT, GOTO 844
%834 IF 'F'.IN.,#GOUT, GOTO 845
%844 CONTINUE

SUBTRACT CORRESPONDING #LV1-LEVEL FROM #LV1-LEVEL.
1111111111111111111111111111
% GOTO #LV3
%846 CONTINUE

ADD CORRESPONDING #LV1-LEVEL TO #LV2-LEVEL.
% GOTO 848
%848 CONTINUE

ADD CORRESPONDING #LV1-LEVEL TO F-LEVEL.
% NOTE: GENERATE COROL STATEMENTS FOR CONTROL
% NOTE: BREAK PROCESSING
%844 #GFIL='BREAKPAR'
% OBEY 102,839
%839 CONTINUE

#PARA-BREAK.
% IF 'R'.IN.,#GOUT, GOTO 851
%852 IF 'O'.IN.,#GOUT, GOTO 853
%854 IF 'P'.IN.,#GOUT, GOTO 855
%856 IF 'N'.IN.,#GOUT, GOTO 857
%858 IF 'M'.IN.,#GOUT, GOTO 861
%861 EXIT.
%868 GOTO 845
%851 #LV1='R'
% #LV2='O'
% #LV3=852
% GOTO 863
% #LV2='P'
% #LV3=853
% GOTO 863
%#855 #LV1='P'
% #LV2='O'
% #LV3=856
% GOTO 863
%#857 #LV1='O'
% #LV2='N'
% #LV3=858
% GOTO 863
%#859 #LV1='N'
% #LV2='M'
% #LV3=862
% GOTO 863
%#861 #LV1='M'
% #LV3=868
%#863 IF #LV1 .NE.'M',GOTO 847
% IF M IN INREC EQUALS M IN CONTROL-BREAK
% GO TO PARA-BREAK-EXIT.
% GOTO 849
%#847 CONTINUE
% IF #LV1 IN INREC EQUALS #LV1 IN CONTROL-BREAK
% GO TO #LV2-BREAK.
%#849 OBEY 870,841
%#841 OBEY 869,842
%#842 OBEY 890,843
%#843 GOTO #LV3
% NOTE: GENERATE PERFORM TOTALS STATEMENTS
%#870 IF 'M'.IN.#GOUT,GOTO 871
%#869 IF 'N'.IN.#GOUT,GOTO 872
%#879 IF 'O'.IN.#GOUT,GOTO 873
%#868 IF 'P'.IN.#GOUT,GOTO 874
%#869 IF 'O'.IN.#GOUT,GOTO 875
%#899 IF 'R'.IN.#GOUT,GOTO 876
% GOTO 877
%#871 CONTINUE
% PERFORM M-TOTALS.
% IF #LV1.EQ.'M',GOTO 877
% GOTO 849
%#872 CONTINUE
% PERFORM N-TOTALS.
% IF #LV1.EQ.'N',GOTO 877
% GOTO 879
%#873 CONTINUE
% PERFORM O-TOTALS.
% IF #LV1.EQ.'O',GOTO 877
% GOTO 888
%#74 CONTINUE
% PERFORM P-TOTALS.
% IF #LV1.EQ.'P',GOTO 877
% GOTO 889
%#75 CONTINUE
% PERFORM Q-TOTALS.
% IF #LV1.EQ.'Q',GOTO 877
% GOTO 889
%#76 CONTINUE
% PERFORM R-TOTALS.
%877 RETURN
%NOTE: GENERATE MORE STATEMENT(S) TO UPDATE CONTROL FIELDS
%881 IF #LV1.EQ.'R',GOTO 881
% IF #LV1.EQ.'O',GOTO 882
% IF #LV1.EQ.'P',GOTO 883
% IF #LV1.EQ.'P',GOTO 884
% IF #LV1.EQ.'N',GOTO 885
% IF #LV1.EQ.'M',GOTO 886
% GOTO 887
%#88 CONTINUE
% MOVE R IN INREC TO R IN CONTROL-BREAK.
%882 CONTINUE
% MOVE Q IN INREC TO Q IN CONTROL-BREAK.
%883 CONTINUE
% MOVE P IN INREC TO P IN CONTROL-BREAK.
%884 CONTINUE
% MOVE Q IN INREC TO Q IN CONTROL-BREAK.
%885 CONTINUE
% MOVE N IN INREC TO N IN CONTROL-BREAK.
%886 CONTINUE
% MOVE M IN INREC TO M IN CONTROL-BREAK.
%887 RETURN
%NOTE: GENERATE PERFORM VERBS FOR CONTROL HEADINGS
%891 IF 'F'.EQ.#LV1,GOTO 891
% IF 'R'.EQ.#LV1,GOTO 892
% IF 'O'.EQ.#LV1,GOTO 893
% IF 'P'.EQ.#LV1,GOTO 894
% IF 'O'.EQ.#LV1,GOTO 895
% IF 'N'.EQ.#LV1,GOTO 896
% IF 'M'.EQ.#LV1,GOTO 897
% GOTO 898
%891 IF 'F'.EQ.#SHED,GOTO 923
% GOTO 892
%892 CONTINUE
PERFORM F-HEAD THRUI F-HEAD-EXIT.

892 IF 'R'.IN.#GHED,GOTO 901
%401 CONTINUE

PERFORM R-HEAD THRUI R-HEAD-EXIT.

893 IF 'O'.IN.#GHED,GOTO 902
%402 CONTINUE

PERFORM Q-HEAD THRUI Q-HEAD-EXIT.

894 IF 'P'.IN.#GHED,GOTO 903
%403 CONTINUE

PERFORM P-HEAD THRUI P-HEAD-EXIT.

895 IF 'Q'.IN.#GHED,GOTO 904
%404 CONTINUE

PERFORM O-HEAD THRUI O-HEAD-EXIT.

896 IF 'N'.IN.#GHED,GOTO 905
%405 CONTINUE

PERFORM N-HEAD THRUI N-HEAD-EXIT.

897 IF 'M'.IN.#GHED,GOTO 906
%406 CONTINUE

PERFORM M-HEAD THRUI M-HEAD-EXIT.

898 IF #LV1.EQ.'M',GOTO 867

GO TO PARA-BREAK-EXIT.

#LV2-BREAK.

899 RETURN
%405 CONTINUE

PARA-BREAK-EXIT. EXIT.

896 CONTINUE

CONTINUE

/1/1//1/1/1/1/1/1/1/1/1/
% NOTE: GENERATE STEERING LINES FOR PART1 MACRO
/1/FILE,0,PART1/
/1/DEF PART1
*IDENTITY #AV2
*COMPILE
*COBOL CARDS (TAPNCRD5)
*OBJECT EDS (ICLA-DEFAULT)
*STANDARD
*CONSOLIDATE EDS XPCK
*SUBGROUPS EDS (SUBGROUPS-RS)
*LOAD

...
*LIST SP
/1/END
/1//1//1//1//1//1//1//1/
% NOTE: COMPLETE GENERATION OF CONSTANTS MACRO
/1/FILE,1,CONSTANTS/
/1/   #GHATCGHE=##GOU1O#GHOED#GOU1O
/1/   #GHATCGOIT=##GOU1O#GOUIT#GOU1O
/1/   #GHATCGV2=##GOU1O#GUV2#GOU1O
/1/   #GHATCGV11=##GOU1O#GUV11#GOU1O
/1/   #GHATCGV12=##GOU1O#GUV12#GOU1O
/1/   #GHATCGS1=##GS01
/1/   #GHATCGS03=##GS03
/1/   #GHATCGS12=##GS12
/1/   #GHATCGS15=##GS15
/1/   #GHATCGVZ=##GUVZ
/1/END
/1//1//1//1//1//1//1//1/
% NOTE: COMPLETE GENERATION OF MACROS
% #GFILE='NOTEPARA'
% OBEY 104,30
% #GFILE='TITLES'
% OBEY 104,31
% #GFILE='TITLEPAR'
% OBEY 104,32
% #GFILE='OTHERPAR'
% OBEY 104,33
% #GFILE='HEADINGS'
% OBEY 104,34
% #GFILE='INPUTREC'
% OBEY 104,35
% #GFILE='CTRLBREAK'
% OBEY 104,36
% #GFILE='SAVETOTL'
% OBEY 104,37
% #GFILE='WRITEPAR'
% OBEY 104,38
% #GFILE='MOVEPARA'
% OBEY 104,39
% #GFILE='OUTPAR'
% OBEY 104,40
% #GFILE='OUTREC'
% OBEY 104,40
% #GFILE='FINALPAR'
% OBEY 104,41
% #GFILE='PAGEPARA'
% OBEY 104,42
% #GFILE='BREAKPAR'
% OBEY 104,43
% #GFILE='ADDPARA'
% OBEY 104,44
% GOTO 9999
11. MESSAGES AND CREATE SUBFILES

CATALOGUE CREATION MACRO.

THIS MACRO CREATES AN EMPTY CATALOGUE AND ESTABLISHES THE SECURITY
SYSTEM FOR PROTECTING ACCESS TO THE FILE DESCRIPTIONS WHICH THE
CATALOGUE WILL EVENTUALLY CONTAIN.

PLEASE TYPE THE PASSWORD TO BE USED BY PERSONS ENTITLED TO AMEND THE
CATALOGUE. THE PASSWORD MAY BE UP TO 8 CHARACTERS LONG AND THE DEFAULT
VALUE IS BLANK. CHARACTERS IN EXCESS OF 8 WILL BE IGNORED.

PLEASE TYPE THE NUMBER OF SECURITY LEVELS BY WHICH ACCESS TO FILE DATA
IS TO BE PROTECTED. THE NUMBER MUST BE GREATER THAN ZERO AND LESS THAN
13. THE DEFAULT REPLY IS 12.
%DEF CREATE
% NOTE: MACRO TO CREATE AN EMPTY CATALOGUE FOR DESCRIPTIONS OF
% NOTE: FILES WITHIN THE DATABASE
% NOTE: IDENTIFY THE MACRO TO THE USER AND REQUEST THE CATALOGUE
% NOTE: PASSWORD
% #LSTRT=1
% #LFIN=7
% CALL 1
% NOTE: READ AND EXTRACT PASSWORD FOR CATALOGUE
% CALL 9
% CALLFSTR(#GPASS,1,8)
% NOTE: REQUEST NO. OF SECURITY LEVELS TO BE USED
% #LSTRT=8
% #LFIN=10
% CALL 1
% NOTE: READ AND VALIDATE SECURITY LEVELS
% CALL 9
% SQUASH
% IF #GSTC.EQ.0,GOTO 3
% IF #GSTC.GT.2,GOTO 4
% NOTE: TEST FIRST CHARACTER FOR DIGIT
% CALLFSTR(#LV1,1,1)
% TESTDIGT(#LV1)
% IF #GTST.NE.0,GOTO 4
% NOTE: TEST SECOND CHARACTER IF PRESENT FOR DIGIT
% IF #GSTC.EQ.1,GOTO 5
% CALLFSTR(#LV1,2,1)
% TESTDIGT(#LV1)
% IF #GSTC.NE.2,GOTO 4
% NOTE: EXTRACT TWO VALID DIGITS
% CALLFSTR(#LV1,1,2)
% NOTE: CONVERT #LV1 TO BINARY
% CALLCONV(#LV1,#LV1)
% IF #LV1.LT.1,GOTO 4
% IF #LV1.GT.12,GOTO 4
% NOTE: SET UP AND WRITE CATALOGUE CONTROL RECORD
% DATE
% TIME
% RESET #S81
% DEPOSIT #S81,#GPASS,#LV1,#GCDT,#GCTM,\n% SELECT CATALOGUE
% REPLACE #S81
% NOTE: WRITE END OF JOB MESSAGE
% DEPOSIT0 EMPTY%CATALOGUE%WITH%#LV1%SECURITY%LEVELS%CREATED]ON\%WRITE
% DEPOSIT #GCDT=#GCTM#AND#READY#FOR#USE,\%WRITE
% GOTO 999
% NOTE: ROUTINE TO PRINT RECORDS FROM THE MESSAGE FILE
% SELECT MESSAGES
% IFREAD #LSTRT,#S81
% DEPOSIT #S81\n% WRITE
% #LSTRT=#LSTRT+1
% IF #LSTRT.LE.#LFIN,GOTO 7
% EXIT
% NOTE: SET DEFAULT NO. OF SECURITY LEVELS TO BE 12
% #LV1=12
% GOTO 6
% NOTE: SECURITY LEVEL ERROR
% WRITE #S81
% DEPOSIT #S81#INVALID#SECURITY#LEVEL#PLEASE#ENTER#CORRECT#VALUE,\%WRITE
% GOTO 2
% NOTE: ISSUE PROMPT AND READ REPLY
% DEPOSIT #S81\n% WRITE
% READ
% EITHER
% 999 CONTINUE
% END
%DEF PASSCHANGE
%NOTE: MACRO TO ALLOW THE CATALOGUE PASSWORD TO BE CHANGED
%NOTE: INTRODUCE MACRO AND REQUEST OLD PASSWORD
DEPOSITO MACRO% TO% CHANGE% CATALOGUE% PASSWORD.
WRITE
DEPOSITO PLEASE% TYPE% THE% OLD% PASSWORD.
WRITE
NOTE: READ AND VALIDATE OLD PASSWORD
CALL 9000
CALLFSTR(#GOLD,1,8)
NOTE: READ CATALOGUE CONTROL RECORD FROM SUBFILE
SELECT CATALOGUE
FREAD 1,#S01
NOTE: EXTRACT FILE AND RECORD COUNTS
READ #S01
CALLUSTR(#LV1,1,1)
#GINP=#GCCT+2
CALLUSTR(#LV2,0,1)
NOTE: EXTRACT PASSWORD
#GINP=#GINP+9
CALLUSTR(#LV3,0,1)
NOTE: EXTRACT NO. OF SECURITY LEVELS
#GINP=#GINP+9
NOTE: COMPARE PASSWORDS
IF #GOLD.EQ.#GOLDC,GOTO 10
NOTE: PASSWORD IN ERROR. TERMINATE RUN.
DEPOSITO #GOLD% IS% AN% INVALID% PASSWORD.% TASK% ABANDONED.
WRITE
GOTO 100
NOTE: ISSUE PROMPT AND READ REPLY
%9000 DEPOSITO :
WRITE
READ
EXIT
NOTE: REQUEST NEW PASSWORD
%10 DEPOSITO PLEASE% TYPE% THE% NEW% PASSWORD% OF% UP% TO% 8% CHARACTERS.
WRITE
NOTE: READ AND EXTRACT NEW PASSWORD
CALL 9000
CALLFSTR(#GPASS,1,8)
NOTE: UPDATE CONTROL RECORD ON CATALOGUE
DATE
TIME
RESET #S01
DEPOSIT1 #LV1,#LV2,#GPASS,#LV3,#GCDT,#GCTM,
REPLACE 1,#S01
NOTE: WRITE JOB COMPLETE MESSAGE
DEPOSITO NEW% PASSWORD% IN% OPERATION% ON% #GCDT% AT% #GCTM.
WRITE
%100 CONTINUE
%END
LIBRARIAN SUBFILE

%DEF LIBRARIAN
% NOTE: #GFCT = COUNT OF FILES IN CATALOGUE
% NOTE: #GRCT = COUNT OF RECORDS IN CATALOGUE
% NOTE: #GREC = CURRENT RECORD POINTER
% NOTE: #S09 = FILE NAME
% NOTE: #GCATP = CATALOGUE PASSWORD
% NOTE: #GPW01 = HIGHEST SECURITY LEVEL PASSWORD FOR FILE
% NOTE: #GFERR = FILENAME ERROR INDICATOR
% NOTE: #GPERR = PASSWORD ERROR INDICATOR
% NOTE: #GNERR = ERROR INDICATOR FOR AN INVALID COBOL NAME
% NOTE: #GSLVL = NO. OF SECURITY LEVELS USED TO PROTECT DATA ACCESS

% NOTE: SELECT CATALOGUE SUBFILE
% SELECT CATALOGUE
% NOTE: READ CATALOGUE CONTROL RECORD INTO #S01
% READ 1,#S01
% NOTE: EXTRACT FILE COUNT, RECORD COUNT, PASSWORD AND NO. OF
% NOTE: SECURITY LEVELS
READ #S01
CALLVSTR(#GFCT,1,,)
#GINP=#GCCT+2
CALLVSTR(#GRCT,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#GCATP,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#GSLVL,0,,)
NOTE: CONVERT NUMERIC FIELDS TO BINARY
CALLCONV(#GFCT,#GFCT)
CALLCONV(#GRCT,#GRCT)
CALLCONV(#GSLVL,#GSLVL)
NOTE: SET UP SECURITY LEVEL CHECK LIST IN #S08
RESET #S08
#LV1=1
105 DEPOSIT8 #LV1,\
#LV1=#LV1+1
IF #LV1.LE.#GSLVL,GOTO 105
NOTE: INTRODUCE MACRO AND REQUEST CATALOGUE PASSWORD
DEPOSIT0 CATALOGUELIBRARIAN.\
WRITE
DEPOSIT0 THIS%MACRO%UPDATES%THE%DATABASE%CATALOGUE%OF%FILE\ CALLCOPY(48,LE%DESCRIPTIONS.)
WRITE
DEPOSIT0 PLEASE%ENTER%THE%CATALOGUE%PASSWORD\ WRITE
NOTE: READ AND VALIDATE PASSWORD
DEPOSIT0 :
WRITE
READ CALLFSTR(#LCATP,1,8)
IF #LCATP.NE.#GCATP,GOTO 101
103 CONTINUE
[CHAIN0,LIBPRELIM]
[LIBPRELIM]
GOTO 999
NOTE: INVALID CATALOGUE PASSWORD
101 DEPOSIT0 #LCATP%IS%AN%INVALID%PASSWORD.%RUN%TERMINATED.\
WRITE
999 CONTINUE
%END
14. LIBPRELIM SUBFILE

%DEF LIBPRELIM

% NOTE: REQUEST OPTION - ADD OR DELETE
%100 DEPOSIT0 DO%YOU%WISH%TO%ADD%OR%DELETE%AN%NEW%FILE%DESCRIPTION%OR%DELETE%AN%NEW%FILE
% CALLCOPY('57,X,ISTING%ONE?)
% WRITE
%104 DEPOSIT0 PLEASE%TYPE%ADD%OR%DELETE,%AT%END%TYPE%FINISH.
% WRITE
% NOTE: READ AND VALIDATE OPTION
% NOTE: OPTION NOT MORE THAN 6 CHARACTERS AND EQUAL TO ADD OR DELETE
% CALL 9000
% SQUASH
% #GOUP=#GSTC
% CALLFSTR(#GOPT,1,0)
% #GOUP=1
% IF #GSTC.GT.6,GOTO 102
% IF #GOPT.EQ.'ADD',GOTO 103
% IF #GOPT.EQ.'DELETE',GOTO 103
% IF #GOPT.EQ.'FINISH',GOTO EXIT
% NOTE: INVALID OPTION
%102 DEPOSIT0 #GOPT%IS%AN%INVALID%RESPONSE
% WRITE
% GOTO 104
% NOTE: REQUEST FILENAME
%103 DEPOSIT0 PLEASE%TYPE%THE%NAME%OF%THE%FILE%WHOSE%DESCRIPTION%YOU
% WRITE
% DEPOSIT0 WISH%TO#GOPT.
% WRITE
% NOTE: READ AND STORE FILENAME
% CALL 9000
% WRITE #599
% NOTE: VALIDATE FILENAME OF UP TO 30 CHARACTERS
% #GLENG=30
% CALL 200
% NOTE: TEST NAME ERROR INDICATOR
% IF #GNERR.EQ.0,GOTO 300
% NOTE: INVALID FILENAME
DEPOSIT TO FILE NAME "%S09%INVALID" FOR REASON LISTED ABOVE
WRITE
GOTO 103
NOTE: VALIDATE NAME STARTING IN POSITION 1 OF INPUT BUFFER
NOTE: #GNERR = ERROR INDICATOR
NOTE: CLEAR ERROR INDICATOR
%200  #GNERR=0
NOTE: TEMPORARILY SAVE CONTENTS OF INPUT BUFFER
WRITE #S01
NOTE: CHECK NO. OF CHARACTERS
SQUASH
IF #GSTC.GT.#GLENG,GOTO 202
NOTE: CHECK FOR IMBEDDED BLANKS
WRITE #S02
IF #S01.NE.#S02,GOTO 203
NOTE: CHECK FIRST CHARACTER FOR HYPHEN OR Z
%210 CALLFSTR(#LV1,1,1)
IF #LV1.IN.'-','GOTO 204
NOTE: CHECK LAST CHARACTER FOR HYPHEN
%211  #GINP=#GSTC
CALLFSTR(#LV1,0,1)
IF #LV1.EQ.'-',GOTO 205
NOTE: SET UP LOOP TO CHECK FOR ALPHABETIC, NUMERIC AND HYPHEN ONLY
%212  #GINP=1
NOTE: ZEROISE COUNT OF ALPHA CHARACTERS
%  #LALPH=0
NOTE: EXTRACT NEXT CHARACTER
%213 CALLFSTR(#LV1,2,1)
NOTE: TEST NUMERIC
TESTDIGT(#LV1)
IF #GST.FQ.0,GOTO 204
NOTE: TEST ALPHABETIC
TESTLETR(#LV1)
IF #GST.FQ.0,GOTO 207
NOTE: TEST HYPHEN
IF #LV1.EQ.'-',GOTO 204
NOTE: INVALID CHARACTER FOUND
DEPOSIT "#LV1"INVALID CHARACTER,% ONLY%ZTO%Z, %ZTO%9%AND%-%ALLOWED
CALLCOPY(54,D)
WRITE
NOTE: SET ERROR INDICATOR ON
#GNERR=1
NOTE: UPDATE POINTER AND TEST FOR END OF LOOP
%206  #GINP=#GINP+1
IF #GINP.GT.#GSTC,GOTO 214
GOTO 213
NOTE: TEST FOR AT LEAST ONE ALPHACHER
%214 IF #LALPH.GT.0,GOTO 208
NOTE: NO ALPHACHER PRESENT
DEPOSIT AT LEAST ONE ALPHABETIC CHARACTER MUST BE INCLUDED
WRITE
#GNERR=1
NOTE: MORE THAN PERMITTED CHARACTERS IN NAME

WRITE
#GNERR=1
GOTO 209

NOTE: BLANKS NOT ALLOWED

WRITE
#GNERR=1
GOTO 210

NOTE: FIRST CHARACTER HYPHEN OR Z

WRITE
#GNERR=1
GOTO 211

NOTE: LAST CHARACTER HYPHEN

WRITE
#GNERR=1
GOTO 212

NOTE: ADD 1 TO ALPHABETIC COUNT

#LALPH=lALPH+1
GOTO 214

NOTE: REQUEST TOP SECURITY LEVEL PASSWORD

WRITE
DEPOSITO #S09

CALL 900

CALLESTR(#GPW01,1,8)

NOTE: SEARCH CATALOGUE FOR FILE

NOTE: #S09 CONTAINS FILENAME
NOTE: #GPW01 CONTAINS 8 CHARACTER PASSWORD
NOTE: #GPERR IS FILENAME ERROR INDICATOR
NOTE: #LPW01 IS FILE PASSWORD FROM CATALOGUE
NOTE: CLEAR ERROR INDICATORS
SELECT CATALOGUE
#GPERR=0
#GFERR=0
NOTE: SET RECORD POINTER FOR FIRST FILE
#REC=2
#LFCT=#GFCT
NOTE: TEST IF ALL FILES SEARCHED
%401 IF #LFCT.EQ.0,GOTO 492
NOTE: READ FILENAME RECORD
FREAD #GREC,#S01
NOTE: EXTRACT FILENAME FROM #S01
VSTR2 #S01,1,,
NOTE: PLACE RECORD IN INPUT BUFFER
READ #S01
NOTE: UPDATE INPUT BUFFER POINTER
#GINP=#GCCT+2
NOTE: EXTRACT FIELD COUNT AND CONVERT TO BINARY
CALLVSTR(#GFDC1,#S01)
CALLCONV(#GFDC1,#GFDC1)
NOTE: TEST FOR REQUIRED FILE
IF #S09.GT.#S02,GOTO 493
IF #S09.LT.#S02,GOTO 492
NOTE: UPDATE INPUT BUFFER POINTER
#GINP=#GINP+#GCCT+1
NOTE: EXTRACT TOP SECURITY LEVEL PASSWORD
CALLFSTR(#LPW01,0,8)
NOTE: TEST FOR VALID PASSWORD
IF #LPW01.EQ.#GPW01,GOTO 301
NOTE: SET INVALID PASSWORD INDICATOR
#GFERR=1
GOTO 301
NOTE: ISSUE PROMPT AND READ REPLY
%9001 DEPOSIT0 :
WRITE
READ
EXIT
NOTE: UPDATE FILE RECORD POINTER
%403 #GREC=#GREC+#GFDC1+#GFDC+3
IF #SRLV.LE.4,GOTO 444
#GREC=#GREC+1
NOTE: REDUCE FILE COUNT
%404 #LFCT=#LFCT-1
GOTO 401
NOTE: SET FILE NAME ERROR INDICATOR ON
%402 #GFERR=1
NOTE: CHAIN TO MACRO SPECIFIED BY USER OPTION
%301 CONTINUE
[CHAINNO,LIB#GOPT]
[LIB#GOPT]
%999 CONTINUE
%END
LIBDELETE SUBFILE

%DEF LIBDELETE
% NOTE: PROCESS DELETE OPTION
% NOTE: DELETE FILE DESCRIPTION FROM CATALOGUE
% NOTE: #GREC IS CATALOGUE FILE RECORD POINTER
% NOTE: #GFCT IS CATALOGUE FILE COUNT
% NOTE: #GCT IS CATALOGUE RECORD COUNT
% NOTE: #GFERR IS FILENAME ERROR INDICATOR SET BY SEARCH
% NOTE: #GPERR IS PASSWORD ERROR INDICATOR SET BY SEARCH
% NOTE: #GFDCT IS COUNT OF FIELDS IN SPECIFIED FILE SET BY SEARCH
% NOTE: #S01 USED AS TEMPORARY STORAGE
% SELECT CATALOGUE
% NOTE: TEST FILENAME ERROR INDICATOR
% IF #GFERR.EQ.1,GOTO 501
% NOTE: TEST PASSWORD ERROR INDICATOR
% IF #GPERR.NE.0,GOTO 502
% NOTE: #LDREC IS NO. OF RECORDS DELETED
% #LDREC=#GFDCT+#GFDCT+3
% IF #GSLVL.LE.4,GOTO 503
% #LDREC=#LDREC+1
% NOTE: DELETE #LDREC RECORDS STARTING AT #GREC
%503 #LV1=1
%506 DELETE #GREC
% #LV1=#LV1+1
% IF #LV1.LE.#LDREC,GOTO 506
% GOTO 505
% NOTE: ERROR FILENAME NOT IN CATALOGUE
%501 DEPOSIT #509%IS%NOT%IN%THE%CATALOGUE
% WRITE
% GOTO 100
% NOTE: PASSWORD ERROR
%502 DEPOSIT #GPW01%IS%NOT%AN%ACCEPTABLE%PASSWORD.%REQUEST%CANCELLED
% WRITE
% GOTO 100
% NOTE: REDUCE FILE COUNT BY
%505 #GFCT=#GFCT-1
% NOTE: REDUCE RECORD COUNT
% #GRC=#GCT-#LDREC
% NOTE: AMEND CATALOGUE CONTROL RECORD
% DATE
% TIME
% RESET #S01
% DEPOSIT #GFCT,#GCT,#GCATP,#GSLVL,#GCDT,#GCTM,
% REPLACE 1,#S01
% DEPOSIT CATALOGUE.UPDATED%ON#$GCDT%AT#$GCTM
% WRITE
%100 CONTINUE
 [CHAIN0,LIBPRELIM]
 [LIBPRELIM]
%END
16. LIBADD SUBFILE

DEFLIBADD
1. NOTE: ADD FILE DESCRIPTION TO CATALOGUE
2. NOTE: #S09 CONTAINS FILENAME
3. NOTE: #GPW1 IS TOP SECURITY LEVEL PASSWORD
4. NOTE: #GREC POINTS AT START OF FILE DESCRIPTION
5. NOTE: #GCT IS NO. OF RECORDS IN CATALOGUE
6. NOTE: #GFCT IS NO. OF FILES IN CATALOGUE
7. NOTE: #GFERR IS FILENAME ERROR INDICATOR
8. NOTE: TEST FOR FILE SAME NAME ALREADY IN CATALOGUE
9. SELECT CATALOGUE
   IF #GFERR.EQ.1,GOTO 700
   NOTE: FILE NAME ALREADY IN CATALOGUE
   DEPOSIT 1, FILE NAME ALREADY IN CATALOGUE
   WRITE
   DEPOSIT 1, "FILE NAME ALREADY IN CATALOGUE"
   WRITE
   2100 CONTINUE,
   [CHAIN#,LIBPRELIM]
   [LIBPRELIM]
   #GTO=999
   NOTE: REQUEST NO. OF FIELDS IN FILE RECORD
   27AA DEPOSIT 1, PLEASE TYPE THE NUMBER OF FIELDS IN A RECORD OF FILE
   WRITE
   NOTE: READ AND VALIDATE FIELD COUNT
   CALL 9999
   NOTE: NOT MORE THAN 4 DIGITS
   #GINP=1
   27AB
   CALLFSTR('#LV1',0,1)
   TESTDIGT('#LV1')
   IF #GTP, #GINP=1
   IF #GTP.LE,#GSTC,GOTO 745
   #GINP=#GSTC
   CALLFSTR('#GFLDS',1,1)
   CALLCONV('#GFLDS,#GFLDS')
   #GINP=1
   IF #GFLDS.LT.1,GOTO 731
   NOTE: BUILD UP FILE RECORD TYPE 1
   RESET #S09
   DEPOSIT 1, #S09,#GFLDS,
   NOTE: REQUEST REMAINING PASSWORDS FOR RECORD TYPE 1
*LSLVL=2  
RESET #S32  
DEPOSIT2 #GPW1,1/
1794 IF #LSLVL.GT.#GSLVL,GOTO 703
2 IF #LSLVL.GT.4,GOTO 703  
DEPOSITO PLEASE TYPE THE PASSWORD FOR SECURITY LEVEL #LSLVL/
2 WRITE
2 CALL 904A  
CALFSTR(#LV1,1,8)  
2 IF '.*'.IN.#LV1,GOTO 702  
2 IF #LV1.IN.#S42,GOTO 702  
2 DEPOSIT2 #LV1,
2 #LSLVL=#LSLVL+1
2 GOTO 744  
744 DEPOSIT1 #S42/
2 NOTE: WRITE FILENAME RECORD TYPE 1 IN CATALOGUE
2 INSERT #GREC,#S41
2 NOTE: UPDATE RECORD POINTER
2 #GREC=#GREC+1
2 GOTO 642
642 NOTE: INVALID NO. OF FIELDS
741 DEPOSITO #GFLS%INVALID NO. OF FIELDS/
2 WRITE
2 GOTO 744  
741 NOTE: DUPLICATE PASSWORD
742 DEPOSITO #LV1%INVALID OR DUPLICATE PASSWORD/
2 WRITE
2 GOTO 744
742 NOTE: TEST FOR MORE THAN 4 SECURITY LEVELS
743 IF #GSLVL.LE.4,GOTO 603
2 GOTO 643
743 NOTE: DUPLICATE PASSWORD
744 DEPOSITO #LV1%INVALID OR DUPLICATE PASSWORD/
2 WRITE
2 GOTO 643
744 NOTE: REQUEST PASSWORDS FOR RECORD TYPE 1C
2 RESET #S42  
2842 IF #LSLVL.GT.#GSLVL,GOTO 802
2 DEPOSITO PLEASE TYPE THE PASSWORD FOR SECURITY LEVEL #LSLVL/
2 WRITE
2 CALL 904A  
CALFSTR(#LV1,1,8)  
2 IF '.*'.IN.#LV1,GOTO 801
2 IF #LV1.IN.#S41,GOTO 801
2 IF #LV1.IN.#S42,GOTO 801  
2 DEPOSIT2 #LV1,
2 #LSLVL=#LSLVL+1
NOTE: WRITE FILENAME RECORD TYPE 1C IN CATALOGUE

NOTE: INSERT #GREC,#S02

NOTE: UPDATE RECORD POINTER

#GREC=#GREC+1

NOTE: READ AND PROCESS FILENAME RECORD TYPE 2

DEPOSIT0 PLEASE TYPE UP TO 770 CHARACTERS OF DESCRIPTION ABOUT FILE

CALLCOPY(#S02)

DEPOSIT0 ON WHICH STORAGE MEDIUM IS THIS FILE?

DEPOSIT0 PLEASE TYPE 2 CHARACTERS, #FOR MAGNETIC TAPE

DEPOSIT0 CR FOR PUNCHED CARDS, #PT FOR PAPER TAPE

DEPOSIT0 ED FOR EXCHANGEABLE DISC STORAGE

NOTE: READ AND VALIDATE FILE MEDIUM

CALL 9000

SQUASH

IF #GSTC,GT,2,GOTO 912

CALLFSTR(#GMED,1,2)

IF #GMED,EQ,"MT",GOTO 901

IF #GMED,EQ,"CR",GOTO 901

IF #GMED,EQ,"PT",GOTO 901

IF #GMED,EQ,"ED",GOTO 901

DEPOSIT0 INVALID FILE MEDIUM. PLEASE RETYPE RESPONSE.

NOTE: S ave FILE MEDIUM

DEPOSIT0 #GMED,

NOTE: WHICH MUST BE UP TO 4 DIGITS WITH VALUE < OR = 2048

DEPOSIT0 PLEASE ENTER BLOCKSIZE

CALL 9000

SQUASH

#GOUP=#GSTC
CALLFSTR(#LV1,0,1)

TESTDIGT(#LV1)

IF #GTST.NE.0,GOTO 902

#GINP=#GINP+1

IF #GINP.LE.#GSTC,GOTO 914

CALLFSTR(#GLKL,1,0)

#GLUP=1

CALLCONV(#GLKL,#GLKL)

IF #GLKL.LT.1,GOTO 902

IF #GLKL.GT.2048,GOTO 932

NOTE: SAVE BLOCK SIZE

DEPOSITI #GLKL,

NOTE: REQUEST AND VALIDATE MAXIMUM RECORD SIZE

DEPOSITI PLEASE TYPE MAXIMUM SIZE OF CHARACTERS IN EACH RECORD

WRITE

CALL #AMM

SWIASH

#GINP=1

CALLFSTR(#LV1,2,1)

TESTDIGT(#LV1)

IF #GTST.NE.0,GOTO 944

#GINP=#GINP+1

IF #GINP.LE.#GSTC,GOTO 945

#GLUP=#GSTC

CALLFSTR(#GRSZ,1,0)

#GINP=1

CALLCONV(#GRSZ,#GRSZ)

IF #GRSZ.LT.1,GOTO 944

IF #GRSZ.GT.#GLKL,GOTO 946

NOTE: SAVE RECORD SIZE

DEPOSITI #GRSZ,

NOTE: REQUEST FILE LABEL

IF #MED.IN.EDT.GOTO 947

NOTE: NULL LABEL ETC

DEPOSITI ,

GOTO 1008

NOTE: INVALID BLOCK SIZE

DEPOSITI INVALID%BLOCKSIZE

WRITE

GOTO 904

NOTE: INVALID RECORD SIZE

DEPOSITI INVALID%RECORD%SIZE

WRITE

GOTO 943

DEPOSITI PLEASE TYPE THE FILE LABEL WHICH MAY BE UP TO 12

WRITE
DEPOSITO CHARACTERS%LONG.%EXCESS%CHARACTERS%WILL%BE%IGNORED.
WRITE
CALL 9000
WRITE #543
FSTR2 #503,1,12
NOTE: SAVE LABEL
DEPOSIT #502,
NOTE: TEST FOR DISC FILE
IF #GTEQ,NE.'ED',GOTO 1006
NOTE: REQUEST ACCESS MODE
DEPOSITO PLEASE%TYPE%ACCESS%MODE%CODE.%SEQ%FOR%SEQUENTIAL
CALLCOPY(#47,SEQFORRANDOM)
WRITE
1000 CALL 9000
SNIASH
IF #GTEQ,#SC,GT.1,GOTO 1413
CALLSTR(#GMODE,1,1)
IF #GNEQ,#SC,1,1,GOTO 1021
1413 DEPOSITO INVALID%ACCESS%MODE
WRITE
GOTO 1000
NOTE: TEST FOR RANDOM ACCESS
1412 IF #GNEQ,#SC,1,GOTO 1002
1411 DEPOSITO $$$,
GOTO 1000
NOTE: REQUEST ORGANISATION METHOD
1410 DEPOSITO PLEASE%TYPE%OR%ORGANISATION%CODE.%OF%DIRECT
CALLCOPY(#49,OF%FOR%INDEXED)
WRITE
1003 CALL 9000
SNIASH
IF #GTEQ,#SC,GT.1,GOTO 1014
CALLSTR(#GORG,1,1)
IF #GNEQ,#SC,0,GOTO 1024
1014 DEPOSITO INVALID%ORGANISATION
WRITE
GOTO 1003
NOTE: REQUEST SYMBOLIC KEY LENGTH
1004 DEPOSITO PLEASE%TYPE%LENGTH%OF%SYMBOLIC%KEY
WRITE
1007 CALL 9000
SNIASH
#GINP=1
1009 CALLFSTR(#LVI,1,1)
TESTDIGT(#LVI)
IF #GTST,NE.0,GOTO 1005
% #GINP= #GINP+1
% IF #GINP.LE.#GSTC,GOTO 1009
% #GINP= #GSTC
% CALLFSTR(#GLSK,1,1)
% #GINP=1
% CALLCONV(#GLSK,#GLSK)
% IF #GLSK,LT.1,GOTO 1045
% IF #GLSK,GT.44,GOTO 1046
% NOTE: SAVE MODE, ORGANISATION AND KEY LENGTH
% DEPOSIT P,#GKEY,#GBAG,#GLSK,
% GOTO 1008
% NOTE: INVALID KEY LENGTH
%1005 DEPOSIT KEY,LENGTH MUST BE IN THE RANGE 1 TO 44
% WRITE
% GOTO 1007
% NOTE: REQUEST NO. OF KEY FIELDS
%1008 DEPOSIT CLOSEDTYPE,#NB,#SSEQ,#SKFIE,
% WRITE
% CALL SQASH
% IF #GSTC.GT.1,GOTO 1024
% CALLFSTR(#GKEY,1,1)
% TESTDIGT(#GKEY)
% IF #STST.NE.0,GOTO 1024
% CALLCONV(#GKEY,#GKEY)
% IF #GKEY,GT.2,GOTO 1014
% NOTE: INVALID NO. OF KEYS
%1024 DEPOSIT NO.,OF KEYS MUST BE IN THE RANGE 1 TO 44
% WRITE
% GOTO 1011
% NOTE: SAVE NO. OF KEYS
%1011 DEPOSIT #GKEY,
% NOTE: WRITE FILE RECORD TYPE 3 TO CATALOGUE
% INSERT #GREC,#S01
% NOTE: UPDATE CATALOGUE RECORD COUNT
% #GREC=#GREC+1
% NOTE: SET UP FIELD KEY CHECK LIST IN #S07
% RESET #S07
% #LVI=0
%1012 DEPOSIT #LVI
% #LVI=#LVI+1
% IF #LVI.LE.#GKEY,GOTO 1012
% RESET #S06
% NOTE: PROCESS FIELD DESCRIPTION RECORDS
% NOTE: SAVE POINTER TO START OF FIELD DESCRIPTION RECORDS
% #FRCST=#GREC
% NOTE: INTRODUCTORY MESSAGE
DEPOSITO YOU ARE ABOUT TO BE ASKED FOR DETAILS OF FIELDS WITHIN

CALLCOPY(S6,A%RECORD)
WRITE
DEPOSITO WHERE POSSIBLE PLEASE ENTER FIELD NAMES IN ASCENDING
CALLCOPY(S3,%START%POSITION)
WRITE
DEPOSITO AS THIS MAKES FOR MORE EFFICIENT PROCESSING
WRITE
NOTE: INITIALISE FIELD COUNT
#LFDS=1
NOTE: SET NAME LENGTH TO 14
#GLEN=14
NOTE: REQUEST AND VALIDATE FIELD NAME
1101 DEPOSITO PLEASE TYPE THE NAME OF FIELD NO. #LFDS
WRITE
1102 CALL 9004
CALL 200
IF #GNERR.EQ.0,GOTO 1103
NOTE: INVALID FIELD NAME
DEPOSITO INVALID FIELD NAME. REASON SPECIFIED ABOVE.
WRITE
GOTO 1102
NOTE: SAVE FIELD NAME AND CHECK FOR DUPLICATE NAME
1103 WRITE #S01
NOTE: #S01 IS FIELD NAME FOR WHICH SCAN IS TO BE MADE
NOTE: #LREC IS RECORD POINTER TO START OF FIELD RECORDS
NOTE: #LFDS IS CURRENT FIELD NO.
NOTE: #LFDS IS NO. OF FIELD TO BE SCANNED
NOTE: INITIALISE FIELD COUNTER
#LFLD=1
#LREC=#LRCST
IF #LFDS.LE.1,GOTO 1106
WRITE
1104 FREAD #LREC,#S42
IF #S01.AT.#S02,GOTO 1105
#LREC=#LREC+2
#LFLO=#LFLO+1
IF #LFLO.LT.#LFDS,GOTO 1104
NOTE: REQUEST SECURITY LEVEL FOR CURRENT FIELD
1106 DEPOSITO WHICH SECURITY LEVEL DO YOU WISH TO ASSIGN TO #S01?
WRITE
DEPOSITO PLEASE TYPE A NUMBER IN THE RANGE 1 TO #GSLUL
CALLCOPY(S4,%HIGHEST%LEVEL)
WRITE
DEPOSITO #GSLUL IS LOWEST LEVEL
NOTE: READ AND VALIDATE SECURITY LEVEL
%1107 CALL 9000
% WRITE #503
% IF "",IN.,#S03,GOTO 1108
% IF #S03,IN.,#S08,GOTO 1108
% NOTE: INVALID SECURITY LEVEL
%1100 DEPOSIT0 #S03% IS% AN% INVALID% SECURITY% LEVEL
% WRITE
% GOTO 1107
% NOTE: DUPLICATE NAME
%1105 DEPOSIT0 DUPLICATE NAME
% WRITE
% GOTO 1102
% NOTE: SAVE SECURITY LEVEL
%1108 DEPOSIT1 ,#503,
% NOTE: REQUEST FIELD START POSITION
DEPOSIT0 IN% WHICH% CHARACTER% POSITION% DOES% THE% FIELD% START?
% WRITE
% DEPOSIT0 % IS% THE% FIRST% CHARACTER% POSITION% IN% THE% RECORD
% WRITE
% NOTE: READ AND VALIDATE START POSITION
%1111 CALL 9000
% SQUASH
% #GINP=1
%1139 CALLFSTR(#LV1,9,1)
% TESTDIGT(#LV1)
% IF #GTST,NE.3,GOTO 1114
% #GINP=#GINP+1
% IF #GINP.LE.#GSTC,GOTO 1109
% #GOUP=#GSTC
% CALLFSTR(#LV1,1,3)
% #GOUP=1
% CALLCONV(#LV1,#LV1)
% IF #LV1.LT.1,GOTO 1114
% IF #LV1.GT.#GRSZ,GOTO 1110
% NOTE: SAVE START POSITION
% DEPOSIT1 #LV1,
% NOTE: REQUEST FIELD LENGTH
DEPOSIT0 HOW% MANY% CHARACTER% POSITIONS% DOES% THE% FIELD% OCCUPY?
% WRITE
% NOTE: READ AND VALIDATE FIELD SIZE
%1114 CALL 9000
% SQUASH
% #GOUP=#GSTC
% #GINP=1
%1113 CALLFSTR(#LV2,0,1)
% TESTDIGT(#LV2)
IF #GTST,NE.0,GOTO 1112
#G1NP=#G1NP+1
IF #G1NP.LE.#GSTC,GOTO 1113
CALLFSTR(#LV2,1,0)
#G01P=1
CALLCONV(#LV2,#LV2)
IF #LV2.LT.1,GOTO 1112
IF #LV2.GT.126,GOTO 1112
#LV3=#LV1+#LV2-1
IF #LV3.GT.#GRSZ,GOTO 1112
NOTE: SAVE FIELD LENGTH
DEPOSIT #LV2,
NOTE: READ AND VALIDATE FIELD TYPE
DEPOSIT0 DOES%THIS%FIELD%CONTAIN%NUMERIC%OR%ALPHANUMERIC%DATA?
WRITE
DEPOSIT0 PLEASETYPE%NUMERIC%OR%ALPHANUMERIC
WRITE

1115 SQUASH
IF #GSTC.NE.1,GOTO 1116
CALLFSTR(#LV1,1,1)
IF #LV1.IN.'AN',GOTO 1200
NOTE: INVALID FIELD TYPE
1116 DEPOSIT0 #LV1INVALID,ONLY%ALPHABETICAL
WRITE
GOTO 1115
NOTE: INVALID START POSITION
11110 DEPOSIT0 STARTPOSITION%MUST%BE%GREATER%THAN%AND%LESS%THAN
WRITE
DEPOSIT0 OR%EQUAL%TO%#GRSZ
WRITE
GOTO 1111
NOTE: INVALID FIELD LENGTH
11112 DEPOSIT0 FIELD%LENGTH%MUST%BE%GREATER%THAN%LESS%THAN%121
WRITE
DEPOSIT0 AND%SUCH%THAT%FIELD%DOES%NOT%OVERFLOW%THE%END%OF%RECORD
WRITE
GOTO 1114
NOTE: TEST FIELD TYPE
1200 IF #LV1.EQ.'A',GOTO 1201
NOTE: REQUEST DECIMAL POINT LOCATION OF NUMERIC FIELD
DEPOSIT0 PLEASETYPE%THE%POSITION%OF%THE%DECIMAL%POINT
WRITE
DEPOSIT0 RELATIVE%TO%THE%RIGHHAND%CHARACTER%OF%THE%FIELD
WRITE
DEPOSIT0 TYPE%LEFT%FOR%RIGHT%FOLLOWED%BY%AN%UNSIGNED%
CALLCOPY(S6, INTEGER)
WRITE
%1202 CALL 9000
SQUASH
CALLFSTR(#LV1,1,1)
IF #LV1,IN.'LR', GOTO 1204
NOTE: INVALID DECIMAL POINT LOCATION
%1203 DEPOSIT0 INVALID%DECIMAL%POINT%LOCATION
WRITE
GOTO 1202
%1204 #GINP=2
%1205 CALLFSTR(#LV1,2,1)
TESTDIGT(#LV1)
IF #STST.NE.0, GOTO 1203
#GINP=#GINP+1
IF #GINP.LE.#GSTC, GOTO 1205
#GINP=#GSTC-1
CALLFSTR(#LV1,2,1)
CALLCONV(#LV1,#LV1)
IF #LV1.GT.99, GOTO 1203
#GINP=#GSTC
CALLFSTR(#LV1,1,1)
#GINP=1
NOTE: STORE POINT LOCATION
DEPOSIT0 #LV1,
GOTO 1206
NOTE: STORE FIELD TYPE FOR ALPHANUMERIC TYPE
%1201 DEPOSIT0 A,
NOTE: REQUEST KEY FIELD INDICATOR
%1202 DEPOSIT0 PLEA$ERTYPE%FIELD%KEY%NO.%IN%THE%RANGE%0$TO9
WRITE
DEPOSIT0 9%DENOTES%NON-KEY%FIELD, 1%DENOTES%THE%MAJOR%KEY
WRITE
%1209 CALL 9000
WRITE #S02
SQUASH
IF #GSTC.GT.1, GOTO 1208
IF #S02,IN.#S07, GOTO 1207
%1208 DEPOSIT0 INVALID%KEY%INDICATOR%OR%FIELD%NOT%OF%LOWEST%SECURI
WRITE
GOTO 1209
%1207 IF #S02,IN.#S06, GOTO 1208
IF #S02,EQ.'0', GOTO 1221
DEPOSIT6 #S02,
%1221 DEPOSIT1 #S02,
IF #S02,EQ.'0', GOTO 1210
NOTE: REQUEST AND ENTER SEQUENCE CODE
DEPOSIT## PLEASE%INDICATE%SEQUENCE%ORDER.%TYPE%A%FOR%ASCENDING%OR%

WRITE
DEPOSIT## D%FOR%DESCENDING
WRITE
%1212 CALL 9000
SQUASH
IF #GSTC,NE.1,GOTO 1211
CALL FSTR(,LV1,1,1)
IF #LV1,IN.'AD',GOTO 1213
NOTE: INVALID SEQUENCE INDICATOR
DEPOSIT## INVALID%SEQUENCE%INDICATOR
WRITE
GOTO 1212
NOTE: SAVE SEQUENCE INDICATOR
DEPOSIT## #LV1,
NOTE: TEST FOR NON-MAJOR KEY
IF #S#2,EQ.'1',GOTO 1214
NOTE: DOES NON-MAJOR KEY FORM A DOMAIN
DEPOSIT## CAN%FIELD%BE%TREATED%AS%A%DOMAIN?.%TYPE%Y%FOR%YES%OR%N%

CALL COPY(5A, FOR%NO)
WRITE
%1216 CALL 9000
SQUASH
IF #GSTC,NE.1,GOTO 1215
CALL FSTR(,LV1,1,1)
IF #LV1,IN.'YN',GOTO 1217
NOTE: INVALID DOMAIN INDICATION
DEPOSIT## INVALID%DOMAIN%INDICATION
WRITE
GOTO 1216
NOTE: DUMMY SEQUENCE
DEPOSIT## ,
GOTO 1230
NOTE: DUMMY DOMAIN INDICATOR
DEPOSIT## ,
GOTO 1218
NOTE: SAVE DOMAIN INDICATOR
DEPOSIT## #LV1,
NOTE: WRITE FIELD DESCRIPTION RECORD TYPE 1
INSERT #GREC,#S01
NOTE: UPDATE RECORD POINTER
#GREC=#GREC+1
NOTE: REQUEST FIELD DESCRIPTION
DEPOSIT## PLEASE%TYPE%UP%TO%7A%CHARACTERS%OF%DESCRIPTION%ABOUT%THE

CALLCOPY(57,%FIELD)
WRITE
CALL 9000
WRITE #S01
INSERT #GREC,#S01
#GREC=#GREC+1
NOTE: UPDATE CURRENT FIELD NO.
#LFLDS=#LFLDS+1
NOTE: DECREASE FIELD COUNT
#GFLDS=#GFLDS-1
IF #GFLDS.GT.0,GOTO 1101
NOTE: CHECK THAT ALL KEYS HAVE BEEN ALLOCATED
#LV1=1
1219  RESET #S01
DEPOSIT1 LV
IF #S01.IN.#S06,GOTO 1220
NOTE: NOT ALL KEYS ALLOCATED
DEPOSIT1 NOT%ALL%KEYS%HAVE%BEEN%ALLOCATED,REQUEST%TERMINATED
WRITE
NOTE: DELETE DESCRIPTION
#LV1=#GSLVL/5
#LV2=#LFLDS-1*2
#LREC=#LV1+#LV2+3
#GREC=#GREC-#LREC
1231  DELETE #GREC
#LREC=#LREC-1
IF #LREC.GT.0,GOTO 1231
GOTO 1290
1220  #LV1=#LW1+1
IF #LV1.LE.#GNKEY,GOTO 1219
NOTE: UPDATE CATALOGUE FILE COUNT AND RECORD COUNT
#GFCF=#GFCF+1
#LV1=#GSLVL/5
#LV2=#LFLDS-1*2
#GRCT=#GRCT+#LV1+#LV2+3
DATE
TIME
RESET #S01
DEPOSIT1 GFCF,#GRCT,#GCATP,#GSLVL,#GCDT,#GCTM,
REPLACE 1,#S01
DEPOSIT1 CATALOGUE%UPDATED%ON%#GCDT%AT%#GCTM
WRITE
GOTO 100
NOTE: VALIDATE NAME STRING STARTING IN 1ST POSITION OF INPUT BUFF
NOTE: #GNERR = ERROR INDICATOR
% NOTE: CLEAR ERROR INDICATOR
%200   #GNERR=0
% NOTE: TEMPORARILY SAVE CONTENTS OF INPUT BUFFER
% WRITE
% NOTE: CHECK NO. OF CHARACTERS KEYED
% SQUASH
% IF #GSTC.GT.#GLENG,GOTO 202
% NOTE: CHECK FOR IMBEDDED BLANKS
%209   WRITE #S02
% IF #S01.NE.#S02,GOTO 203
% NOTE: CHECK FIRST CHARACTER FOR HYPHEN OR Z
%210   CALFSTR(#LV1,1,1)
% IF #LV1.IN.'-Z',GOTO 204
% NOTE: CHECK LAST CHARACTER FOR HYPHEN
%211   #GINP=#GSTC
% CALLFSTR(#LV1,0,1)
% IF #LV1.EQ.'-',GOTO 205
% NOTE: SET UP LOOP TO CHECK FOR ALPHABETIC, NUMERIC AND HYPHEN ONLY
%212   #GINP=1
% NOTE: ZEROISE COUNT OF ALPHA CHARACTERS
% #LALPH=0
% NOTE: EXTRACT NEXT CHARACTER
%213   CALFSTR(#LV1,0,1)
% NOTE: TESTDIG(#LV1)
% IF #GTST.EQ.0,GOTO 206
% NOTE: TEST ALPHABETIC
% TESTLETR(#LV1)
% IF #GTST.EQ.0,GOTO 207
% NOTE: TEST FOR HYPHEN
% IF #LV1.EQ.'-',GOTO 206
% NOTE: INVALID CHARACTER FOUND
% DEPOSIT #LV1#INVALID#CHARACTER,%ONLY%LETORZ,%0%TO%9%AND-%ALLOWE
% CALLCOPY(54,H)
% WRITE
% NOTE: SET ERROR INDICATOR ON
% #GNERR=1
% NOTE: UPDATE POINTER AND TEST FOR END OF LOOP
%206   #GINP=#GINP+1
% IF #GINP.GT.#GSTC,GOTO 214
% GOTO 213
% NOTE: TEST FOR AT LEAST ONE ALPHA CHARACTER
%214   IF #LALPH.GT.0,GOTO 208
% NOTE: NO ALPHA CHARACTER CHARACTER PRESENT
% DEPOSIT AT LEAST ONE ALPHABETIC CHARACTER MUSBE INCLUDED
% WRITE
% #GNERR=1
%208 EXIT
% NOTE: MORE THAN PERMITTED CHARACTERS IN NAME
%202 DEPOSIT0 MORE%THAN%#LEN%CHARACTERS%IN%NAME\n% WRITE
% #GNERR=1
% GOTO 209
% NOTE: BLANKS NOT ALLOWED
%203 DEPOSIT0 BLANKS%NOT%ALLOWED\n% WRITE
% #GNERR=1
% GOTO 210
% NOTE: FIRST CHARACTER HYPHEN OR Z
%204 DEPOSIT0 Z%OR%HYPHEN%NOT%ALLOWED%AS%FIRST%CHARACTER\n% WRITE
% #GNERR=1
% GOTO 211
% NOTE: LAST CHARACTER HYPHEN
%205 DEPOSIT0 HYPHEN%NOT%ALLOWED%AS%LAST%CHARACTER\n% WRITE
% #GNERR=1
% GOTO 212
% NOTE: ADD 1 TO ALPHABETIC COUNT
%207 #LALPH=#LALPH+1
% GOTO 206
% NOTE: ISSUE PROMPT AND READ REPLY
%9400 DEPOSIT0 :
% WRITE
% READ
% EXIT
%999 CONTINUE
%END
17. DIAL AND STAGE SUPPLIES

Stage 1 - Introduction
This system is a self-teaching COBOL program generator. By means of a question and answer dialogue the system will enable you to
specify both your problem and the data to be used. The system will then
generate a COBOL program to provide you with your answers.
If at any time during the dialogue you wish to change your mind about an
earlier decision or require assistance please type HELP, should you wish to
abandon the session please type CTRL A instead.
In many instances the system offers you a default response to a
question, to take advantage of this option you simply press the accept
key on your terminal keyboard.
The system takes a relational view of the database.
The database is a collection of stored data available for use in the
solution of a problem.
The database is made up of domains, each of which contains a set of data
items relating to a particular attribute.
E.g. an employee-name domain contains the set of all employee names and
a department-no domain contains the set of all department numbers.
A different number of items may be present in each domain set.
A relation may be defined on several domains such that for every item
in one domain of the relation there exists one and only one
corresponding item in each of the other domains in the relation.
E.g. a personnel relation may be defined on the domains employee-name,
address and department-no, thus each employee belongs to only one
department and lives at one address, although several employees
may belong to the same department or live at the same address.
The easiest way to visualise the data in a relation is in tabular form,
where the domain names provide the column headings and the items from
each domain set provide the row entries.
This the data in the personnel relation mentioned above could be
considered as a table of which the following is an extract:

<table>
<thead>
<tr>
<th>EMPLOYEE-NAME</th>
<th>ADDRESS</th>
<th>DEPARTMENT-NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWN A.</td>
<td>124 HIGH STREET, SOUTHON</td>
<td>14</td>
</tr>
<tr>
<td>JONES T.</td>
<td>16 CHURCH ROAD, EASTLEY</td>
<td>92</td>
</tr>
</tbody>
</table>

The generated COBOL program will retrieve your data in such a way that
you may consider the data in the relation table to be available on a
row by row basis, one row at a time.

User stage:

*NOTE: INTRODUCTION - CHECK POINT 1
*NOTE: SET CHECK POINT NO.
*#GCKPT=1
*NOTE: PRINT INTRODUCTORY DIALOGUE FOR STAGE 1
*SELECT DIAL
*#LINE=1
*F450 #LINE, #SAI
*DEPOSIT #SAI
*WRITE
*IF #LINE, LE, +44, GOTO 1
*NOTE: CHAIN TO STAGE2
(STRING, STAGE2)
*END
STAGE 2 - PASSWORD DIALOGUE
ACCESS TO THE SYSTEM DATABASE IS BY MEANS OF A RELATION NAME AND ITS
ASSOCIATED PASSWORD. THE RELATION NAMES AND PASSWORDS ARE ALLOCATED BY
THE DATABASE ADMINISTRATOR TO WHOM AN APPLICATION TO USE THE SYSTEM
SHOULD BE MADE. DATA FROM MORE THAN ONE RELATION MAY BE REQUESTED TO
SOLVE YOUR PROBLEM.

```sql
%DEF STAGE2#
% NOTE: PASSWORD DIALOGUE - CHECK POINT 2
% NOTE: SET CHECK POINT NO.
% #GCKPT=2
% NOTE: PRINT INTRODUCTORY DIALOGUE FOR STAGE 2
% SELECT DIAL2
% #LLINE=1
% FREAD #LLINE,#S41
% DEPOTS #S01
% WRITE
% #LLINE=#LLINE+1
% IF #LLINE.LE.5,GOTO 1
% NOTE: ZEROISE RELATION COUNT AND RECORD POINTER
% #GNREL=0
% #LREC=0
% NOTE: SELECT RELATION DICTIONARY
% SELECT RELATIONDICT
% #LITEM='FIRST'
% NOTE: REQUEST RELATION NAME
% DEPOTS PLEASE%TYPE%THE%NAME%OF%THE%#LITEM%RELATION%WHOSE%DATA
% WRITE
% DEPOTS YOU%WISH%TO%ACCESS
% WRITE
```
NOTE: READ AND SAVE RELATION NAME
CALL 9001
WRITE #S01
NOTE: REQUEST PASSWORD
DEPOSITI PLEASE TYPE THE PASSWORD FOR RELATION #S01
WRITE
NOTE: READ AND SAVE PASSWORD
CALL 9001
CALLFSTR(#LPASS,1,4)
IF #GNREL.NE.4, GOTO 8
NOTE: INCREMENT RELATION COUNT AND RECORD POINTER
#LREC=#LREC+1
#GNREL=#GNREL+1
NOTE: ENTER RELATION NAME AND PASSWORD IN RELATION DICTIONARY
DEPOSITI #LPASS,
INSERT #LREC,#S01
IF #GNREL.EQ.4, GOTO 7
DEPOSITI HAVE YOU ANY MORE RELATIONS TO ENTER? PLEASE TYPE
CALLCOPY(S1,YE FOR YES OR NO)
WRITE
DEPOSITI FOR YES, THE DEFAULT REPLY IS "YES"
WRITE
CALL 9001
CALLFSTR(S1,1,1)
SQUASH
IF #STC.GT.1, GOTO 12
IF #STC.EQ.4, GOTO 2
IF #STC.LT.1, GOTO 3
NOTE: INVALID RESPONSE
WRITE
GOTO 4
NOTE: MORE RELATIONS TO CODE
#LITE='NEXT'
GOTO 4
NOTE: NO MORE RELATIONS TO ENTER CHAIN TO NEXT STAGE
CONTINUE
CHAIN(STAGE3)
(STAGE3#1)
GOTO 9999
NOTE: ISSUE PROMPT, READ RESPONSE AND DETECT PLEA FOR HELP
DEPOSITI :
WRITE
READ
CALLFSTR(S1,1,5)
IF #LV1.EQ.'HELP',GOTO 9001
EXIT
9001 CONTINUE
CHAIN HELP
HELP
GOTO 9999
NOTE: CHECK FOR DUPLICATE RELATION NAME
NOTE: INITIALISE RECORD POINTER
#LREC=1
NOTE: READ AND EXTRACT RELATION NAME
FREAD #LREC,#S02
VSTR3 #S02,1,,
NOTE: COMPARE RELATION NAMES
IF #S01.EQ.#S03,GOTO 11
IF #S01.LT.#S03,GOTO 9
NOTE: TEST FOR ANY MORE RELATIONS TO SCAN
IF #GNREL.EQ.#LREC,GOTO 7
NOTE: UPDATE RECORD POINTER
#LREC=#LREC+1
GOTO 10
NOTE: DUPLICATE RELATION NAME
DEPOSIT DUPLICATE RELATION NAME,NAME IGNORED
WRITE
GOTO 4
9999 CONTINUE
END
19. DIAL3 AND STAGE3 SUBFILES

STAGE 3 - USERS DATABASE SIM:MODEL
E.G. ITEM SIZE POINT LOCATION ITEM CONTENTS NUMERIC VALUE OF CONTENTS

<table>
<thead>
<tr>
<th>Item Size</th>
<th>Point Location</th>
<th>Item Contents</th>
<th>Numeric Value of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>L2</td>
<td>17463</td>
<td>174.63</td>
</tr>
<tr>
<td>3</td>
<td>R3</td>
<td>468</td>
<td>468000</td>
</tr>
</tbody>
</table>

WHERE DATA FROM A RELATION IS READILY AVAILABLE IN AN ORDERED SEQUENCE, THE KEY DOMAINS WHICH DETERMINE THAT SEQUENCE WILL ALSO BE LISTED IN ORDER, STARTING WITH THE MAJOR KEY (KEY NO. 1) AND FINISHING WITH THE MINOR KEY.

%DEF STAGE3##
% NOTE: DATABASE SIM:MODEL - CHECK POINT 3
% NOTE: SET CHECKPOINT NO.
% #GCKPT=3
% NOTE: PRINT INTRODUCTORY DIALOGUE
#LLINE=1
SELECT DIAL3
FREAD #LLINE,#S01
DEPOSIT # #S01
WRITE
#LLINE=#LLINE+1
IF #LLINE.LE.16,GOT 1
% NOTE: SELECT: CATALOGUE, READ: CONTROL RECORD, EXTRACT FIELDS
% NOTE: AND CONVERT NUMERIC FIELDS TO BINARY
SELECT CATALOGUE
READ #S01
NOTE: CATALOGUE FILE COUNT
CALLVSTR(#LCFCT,1,,)
#GINP=#GCCT+2
NOTE: CATALOGUE RECORD COUNT
CALLVSTR(#LCRCT,0,,)
#GINP=#GINP/#GCCT+1
NOTE: CATALOGUE PASSWORD
CALLVSTR(#LCPW,0,,)
#GINP=#GINP/#GCCT+1
NOTE: NO. OF SECURITY LEVELS
CALLVSTR(#LSLVL,0,,)
CALLCONV(#LCFCT,#LCFCT)
CALLCONV(#LCRCT,#LCRCT)
CALLCONV(#LSLVL,#LSLVL)
NOTE: INITIALISE SUBMODEL SUBFILE COUNTERS - FILE AND RECORD,
#LSFCT=0
#LSRCT=1
NOTE: ZEROISE DOMAIN COUNT
#LDCT=0
NOTE: INITIALISE RELATION DICTIONARY POINTER
#LRPT=1
NOTE: INITIALISE CATALOGUE POINTER
#LCPT=2
NOTE: CLEAR ERROR INDICATOR
#LERR=0
NOTE: SELECT RELATION DICTIONARY
NOTE: READ RELATION RECORD, EXTRACT RELATION NAME AND PASSWORD
CALL S0
NOTE: READ FILENAME RECORD TYPE 1 INTO #S01
FREAD #LCPT,#S01
NOTE: EXTRACT FIELDS
NOTE: FILENAME INTO #S03
VSTR3 #S01,1,,
NOTE: EXTRACT FIELD COUNT AND CONVERT TO BINARY
READ #S01
#GINP=#GCCT+2
CALLVSTR(#LCFDC,0,,)
CALLCONV(#LCFDC,#LCFDC)
NOTE: SAVE POINTER TO START OF PASSWORDS
#LINP=#GINP/#GCCT+1
NOTE: TEST FOR MORE THAN 4 SECURITY LEVELS
IF #LSLVL.LE.4,GOTO 4
NOTE: READ REMAINING PASSWORDS INTO #S04
#LCPT=#LCPT+1
NOTE: COMPARE RELATION NAME WITH CATALOGUE FILENAME

IF #S02.GT.#S03,GOTO 6
IF #S02.LT.#S03,GOTO 7
NOTE: MATCH FOUND - PRINT RELATION NAME AND DOMAIN HEADINGS
WRITE
DEPOSITO RELATION%NAME%#S02

WRITE
DEPOSITOR DETAIL%OF%DOMAINS%WITHIN%THE%RELATION
WRITE
DEPOSITOR DOMAIN%NAME%ITEM%SIZE%TYPE%POINT%LOCATION
WRITE
NOTE: DETERMINE PASSWORD SECURITY LEVEL

#LVI=1
#GINP=#LINP
CALLFSTR(#LPA**S,0,8)
#GINP=#GINP+9
IF #LPA**S.EQ.#LRPW,GOTO 10
#LVI=#LVI+1
IF #LVI.GT.#LSLVL,GOTO 11
IF #LVI.LE.4,GOTO 8
NOTE: LOAD REMAINING PASSWORDS
READ #S04
#GINP=1
CALLFSTR(#LPA**S,0,8)
#GINP=#GINP+9
IF #LPA**S.EQ.#LRPW,GOTO 10
#LVI=#LVI+1
IF #LVI.LE.4,GOTO 12
NOTE: SET ERROR INDICATOR AND PRINT MESSAGE
#LERR=1
DEPOSITOR INVALID%PASSWORD%NO%DOMAINS%ACCESSIBLE
WRITE
NOTE: UPDATE CATALOGUE POINTER
#LCPT=#LCPT+3+#LCFDC+#LCFDC
GOTO 42
NOTE: UPDATE RELATION POINTER
#LRPT=#LRPT+1
NOTE: TEST FOR MORE RELATIONS
IF #LRPT.GT.#GNREL,GOTO 14
NOTE: SELECT RELATION DICTIONARY AND PROCESS NEXT RELATION RECORD
CALL 50
GOTO 4
NOTE: UPDATE CATALOGUE POINTER
#LCPT=#LCPT+3+#LCFDC+#LCFDC
IF #LSLVL.LE.4,GOTO 16
NOTE: TEST FOR END OF CATALOGUE

IF #LCPT.LT.#LCRCT,GOTO 3
NOTE: RELATION NOT IN CATALOGUE
#LERR=1
DEPOSIT0 RELATION%$O2%NOT%IN%DATABASE
WRITE
GOTO 14
#LERR=1
DEPOSIT0 RELATION%$O2%NOT%IN%DATABASE
WRITE
GOTO 13
NOTE: TEST ERROR INDICATOR
IF #LERR.EQ.0,GOTO 17
DEPOSIT0 USER%ERRORS%PREVENT%FURTHER%PROGRESS.%DO%YOU%WISHT0%
CALLCOPY(54,RESPECT?YOUR)
WRITE
DEPOSIT0 RELATIONS?PLEASE%TYPE%Y%FOR%YES%OR%N%FOR%NO.THE%DEFAULT
CALLCOPY(57,%REPLY%IS%NO.)
WRITE
CALL 9000
SQUASH
IF #GSTC.GT.1,GOTO 19
CALLFSTR(#LV1,1,1)
IF #LV1.EQ.'Y',GOTO 20
IF #LV1.IN.'N',GOTO 21
DEPOSIT0 INVALID%RESPONSE
WRITE
GOTO 18
NOTE: WRITE SUBMODEL CONTROL RECORD
SELECT SUBMODEL
RESET #S01
DEPOSIT1 #LSFCT,#LSRCT,
REPLACE 1,#S01
NOTE: WRITE DOMAINDICT CONTROL RECORD
SELECT DOMAINDICT
REPLACE 1,#LDICT
[CHAIN0,STAGE3-1]
[STAGE3-1@1@]
GOTO 9999
CONTINUE
[CHAIN0,STAGE2]
[STAGE2@1@]
GOTO 9999
CONTINUE
WRITE
GOTO 9999
NOTE: ROUTINE TO READ USERS RESPONSE AND DETECT A PLEA FOR HELP
NOTE: ISSUE PROMPT
9000 DEPOSIT0 :\nWRITE
READ
CALLFSTR(#LV1,1,5)
IF #LV1.EQ.'HELP',GOTO 9001
EXIT
9001 CONTINUE
[CHAIN0,HELP]
[HELP018]
GOTO 9999
NOTE: EXTRACT FILE RECORD TYPE 3 FROM CATALOGUE
10
#LCPT=#LCPT+2
FREAD #LCPT,#S03
NOTE: INITIALISE KEY DICTIONARY COUNT
#GKEY=0
NOTE: INITIALISE SUBMODEL FIELD COUNT
#LSFDC=0
NOTE: SELECT SUBMODEL AND INSERT TYPE 1 AND TYPE 3 RECORDS
SELECT SUBMODEL
#LSRCT=#LSRCT+1
INSERT #LSRCT,#S02
NOTE: SAVE RECORD POINTER
#LSPT=#LSRCT
#LSRCT=#LSRCT+1
INSERT #LSRCT,#S03
NOTE: ADD 1 TO SUBMODEL FILE COUNT
#LSFCT=#LSFCT+1
NOTE: SELECT CATALOGUE AND READ TYPE 1 RECORD
SELECT CATALOGUE
#LCPT=#LCPT+1
FREAD #LCPT,#S01
NOTE: REDUCE CATALOGUE FIELD COUNT BY 1
#LCFDC=#LCFDC-1
NOTE: EXTRACT NAME AND SECURITY LEVEL - NAME IN #S02
VSTR2 #S01,1,0
READ #S01
#GINP=#GCCT+2
CALLFSTR(#LV2,0,1)
CALLC0NVC(#LV2,#LV2)
NOTE: TEST IF USER HAS ACCESS TO FIELD
NOTE: USER HAS ACCESS, EXTRACT REMAINING FIELDS

CALLVSTR(#LV3,0,,)
#GINP=#GINP+2
#GINP=#GINP+GCCT+1
CALLVSTR(#LV4,0,,)
#GINP=#GINP+GCCT+1
CALLVSTR(#LV5,0,,)
#GINP=#GINP+GCCT+1
CALLVSTR(#LV6,0,,)
#GINP=#GINP+GCCT+1
CALLVSTR(#LV7,0,,)
#GINP=#GINP+GCCT+1
CALLVSTR(#LV8,0,,)
#GINP=#GINP+GCCT+1
CALLVSTR(#LV9,0,,)

NOTE: SELECT DOMAIN DICTIONARY AND BINARY SEARCH FOR NAME
SELECT DOMAINDICT

#LLow=2
#Lhigh=#LDCT+1

IF #LLow.GT.#Lhigh,GOTO 24
#LCUR=#LLow+#Lhigh
#LCUR=#LCUR/2
FREAD #LCUR,#S04

NOTE: EXTRACT DOMAIN NAME IN #S03
VSTR3 #S04,1,,
IF #S03.GT.#S02,GOTO 25
IF #S03.EQ.#S02,GOTO 26
#LLow=#LCUR+1
GOTO 27
#Lhigh=#LCUR-1
GOTO 27

NOTE: MATCHING NAME FOUND - TEST FOR KEY DOMAIN
26 IF #LV7.NE.'0',GOTO 23
NOTE: DOMAIN NAME IN MORE THAN ONE RELATION - RESOLVE AMBIGUITY BY

NOTE: APPENDING RELATION COUNT TO FILL OUT NAME
243 DEPOSIT2 #LRPT
CALL LENV #S02
IF #GSTC.LT.15,GOTO 43
GOTO 44
NOTE: MATCHING NAME NOT FOUND - TEST FOR KEY DOMAIN
224 IF #LV7.EQ.'0',GOTO 28
NOTE: ENTER DOMAIN NAME IN KEY DICTIONARY
SELECT KEYDICT
#GNKEY=#0NKEY+1
#LDCT=#LDCT+1
#LSRCT=#LSRCT+1
#LSFDC=#LSFDC+1
#COUP=1
#GOUP=2
#GOUP=3
#GOUP=38
#LCPT=#LCPT+1
#LCPT=#LCPT+1
#LSFDC=1
#LSPT,#S01
#LSFDC,#S01
NOTE: ENTER NAME IN DOMAIN DICTIONARY
NOTE: SELECT SUBMODEL AND ENTER FIELD DESCRIPTION
NOTE: SELECT SUBMODEL
NOTE: SELECT SUBMODEL
NOTE: TEST FOR MORE FIELDS
NOTE: UPDATE CATALOGUE POINTER
NOTE: UPDATE SUBMODEL FILE RECORD TYPE
NOTE: PRINT KEY HEADING
NOTE: SELECT KEYDICT
% #LKPT=1
%41 FREAD #LKPT,#S01
#GOUP=3
DEPOSIT #LKPT\n#GOUP=10
DEPOSIT #S01\nWRITE
#LKPT=#LKPT+1
IF #LKPT.LE.#GNKEY,GOTO 41
NOTE: UPDATE RELATION POINTER AND TEST FOR MORE RELATIONS
%42 #LRPT=#LRPT+1
IF #LRPT.GT.#GNREL,GOTO 14
NOTE: READ AND EXTRACT DETAILS OF NEXT RELATION
CALL 50
GOTO 16
NOTE: READ RELATION RECORD, EXTRACT NAME AND PASSWORD
%50 SELECT RELATIONDICT.
% FREAD #LRPT,#S01
VSTR2 #S01,1,,
READ #S01
#GINP=#GCCT+2
CALLUSTR(#LRPW,0,,)
SELECT CATALOGUE
EXIT
%9999 CONTINUE
%END
20. STAGE3-1 SUBFILE

%DEF STAGE3-1@@
% NOTE: CHECK FOR DISSJOINTED OR DISCONNECTED RELATIONS
% NOTE: TEST FOR MORE THAN ONE RELATION
% IF #GNREL.EQ.1,GOTO 23
% NOTE: SELECT SUBMODEL, READ AND EXTRACT FILE AND RECORD COUNTS
% SELECT SUBMODEL
% READ 1,#S01
% CALLVSTR(#LSRCT,0,,)
% NOTE: CONVERT COUNTS TO BINARY
% CALLCONV(#LSRCT,#LSRCT)
% NOTE: INITIALISE SUBMODEL RECORD POINTER
% #LSPT=2
% NOTE: ZEROISE KEYCOUNT
% #GNKEY=0
% NOTE: READ FILE DESCRIPTION RECORDS
% CALL 8
% NOTE: TEST KEY COUNT
% IF #LKYCT.GT.#GNKEY,GOTO 2
% NOTE: UPDATE RECORD POINTER
% #LSPT=#LSPT+#LFDCT+1
% NOTE: TEST FOR END OF SUBMODEL
% IF #LSPT.LE.#LSRCT,GOTO 1
% GOTO 7
% NOTE: UPDATE KEY COUNT
% #GNKEY=#LKYCT
% NOTE: SAVE FILE POINTER
% #LSAVE=#LSPT-1
% NOTE: SET SUBMODEL RECORD POINTER
% #LSPT=#LSPT+1
% NOTE: READ FIELD RECORD
% CALL 9
% NOTE: TEST FOR KEY FIELD
% IF #LKEY.NE.'0',GOTO 5
% NOTE: REDUCE FIELD COUNT
% #LFDCT=#LFDCT-1
% NOTE: UPDATE RECORD POINTER
NOTE: TEST FOR END OF FIELD RECORDS
IF #LFDCT.NE.0,GOTO 4
GOTO 3

NOTE: SELECT KEY DICTIONARY AND ENTER KEY NAME
CALLCONV(#LKEY,#LKEY)
SELECT KEYDICT
REPLACE #LKEY,#S01
NOTE: SELECT SUBMODEL
SELECT SUBMODEL
GOTO 6

NOTE: RESET SUBMODEL RECORD POINTER
#LSPT=1
NOTE: READ FILE DESCRIPTION RECORDS
#LSPT=#LSPT+1
NOTE: TEST FOR SAVED FILE
IF #LSAVE.EQ.#LSPT,GOTO 24
NOTE: READ FILE RECORDS
CALL 8
NOTE: READ FIELD RECORD
#LSPT=#LSPT+1
CALL 9
NOTE: TEST FOR KEY FIELD
IF #LKEY.NE.'0',GOTO 12
NOTE: REDUCE FIELD COUNT
#LFDCT=#LFDCT-1
IF #LFDCT.EQ.0,GOTO 13
GOTO 10

NOTE: CONVTRT KEY NO. TO BINARY AND SAVE
CALLCONV(#LKEY,#LKEY)
#LSKY=#LKEY
NOTE: BINARY SEARCH KEY DICTIONARY FOR MATCHING NAME
#LLow=1
#LHigh=#GNKEY
SELECT KEYDICT
IF #LLow.GT.#LHigh,GOTO 13
#LCUR=#LHigh+$LLow
#LCUR=#LCUR/2
FREAD #LCUR,#S02
IF #S02.EQ.#S01,GOTO 14
IF #S02.LT.#S01,GOTO 15
#LHigh=#LCUR-1
GOTO 16
#LLow=#LCUR+1
GOTO 16
NOTE: REDUCE FIELD COUNT
GOTO 14
#LFDCT=#LFDCT-1
NOTE: UPDATE RECORD POINTER

\#LSPT=\#LSPT+1

NOTE: TEST FOR MORE FILE DESCRIPTIONS

\#LSPT=\#LSPT+\#LSRCT+1

NOTE: CHAIN TO NEXT STAGE

\#LSPT=\#LSPT+\#FDC+F2

NOTE: UPDATE SUBFILE POINTER

\#LSPT=\#LSPT+\#FDC+F2

NOTE: ERROR DETECTED

DEPOSIT YOUR CHOSEN RELATIONS CANNOT BE JOINED TOGETHER. DO YOU

CALLCOPY(56,\%WISH\%TO)

WRITE

DEPOSIT RESPECIFY YOUR RELATION NAMES? PLEASE TYPE Y FOR YES OR

CALLCOPY(57,\%NOTWISH\%THE)

WRITE

DEPOSIT DEFAULT REPLY IS NO.

WRITE

CAL 9000

SQUASH

IF \#GSTC.GT.1, GOTO 19

CALLFSTR(\#LVI,1,1)

IF \#LVI.EQ.'Y', GOTO 20

IF \#LVI.IN.'N', GOTO 21

DEPOSIT INVALID RESPONSES

WRITE

GOTO 22

CONTINUE

[CHAIN, STAGE2]

GOTO 9999

GOTO 9999

DEPOSIT RUN ABANDONED

WRITE

GOTO 9999

NOTE: ROUTINE TO READ USERS RESPONSE AND DETECT A PLEA FOR HELP

READ

CALLFSTR(\#LVI,1,5)

IF \#LVI.EQ.'HELP', GOTO 9001

EXIT

CONTINUE

[CHAIN, HELP]
NOTE: SELECT SIM-MODEL AND UPDATE RECORD POINTER

SELECT SIM-MODEL
#LSPT=#LSPT+1
CALL 9

IF #LKEY, 'F0', 'A', GOTO 14
NOTE: CONVERT KEY NO. TO BINARY
CALL CONV(#LKEY, #LKEY)

NOTE: CALCULATE KEY LOCATION IN DICTIONARY
NOTE: CHECK FOR VALID POINTER
#LKPT=#LLOW-#LSKY+#LKEY
IF #LKPT LE A, GOTO 13
IF #LKPT GT #GKEY, GOTO 13
NOTE: SELECT KEY DICTIONARY AND CHECK KEY NAME
SELECT KEYDICT
FREAD #LKPT, #S42
IF #S41 NE #S42, GOTO 13
GOTO 14

NOTE: READ FILE DESCRIPTION RECORDS TYPE 1 AND TYPE 3
FREAD #LSPT, #S41
NOTE: EXTRACT FIELD COUNT
VSTR2 #S41, 1,,
READ #S41
#GINP=#GCCT+2
CALL VSTR(#LFDCT, 0,,)
NOTE: READ TYPE 2 RECORD
#LSPT=#LSPT+1
FREAD #LSPT, #S41
NOTE: EXTRACT KEY COUNT
READ #S41
CALL VSTR(#LV1, 1,,)
#GINP=#GCCT+2
CALL VSTR(#LV2, 0,,)
#GINP=#GINP+#GCCT+1
CALL VSTR(#LV3, 0,,)
#GINP=#GINP+#GCCT+14
CALL VSTR(#LV4, 0,,)
#GINP=#GINP+#GCCT+1
CALL VSTR(#LV5, 0,,)
#GINP=#GINP+#GCCT+1
CALL VSTR(#LV6, 0,,)
#GINP=#GINP+#GCCT+1
CALL VSTR(#LKYCT, 0,,)
NOTE: CONVERT KEY COUNT TO BINARY
CALL CONV(#LKYCT, #LKYCT)
EXIT
NOTE: READ FIELD DESCRIPTION RECORD
FREAD #LSPT,#S02
NOTE: EXTRACT FIELD NAME AND KEY NO.
VSTR1 #S02,1,,
READ #S02
#GINP=#GCCT+2
CALLVSTR(#LV1,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LV2,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LV3,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LV4,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LW5,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LKEY,,)
EXIT
99999 CONTINUE
END
21. DIAL4 AND STAGE4 SUBFILES

STAGE 4 - SELECTION OF RELEVANT DOMAINS

Your database contains a number of domains not all of which may be required to solve your current problem. The system is about to list the contents of your database domain by domain; after each domain name you will be asked whether or not the current domain features in your problem. Should you so desire, the system will automatically accumulate a total for the items retrieved from any numeric domain. You will be asked to indicate if you wish to avail yourself of this feature.

DEF STAGE4#

* NOTE: CHECK POINT 4 - SELECTION OF PROBLEM DOMAINS
* NOTE: SET CHECK POINT NO.
* #NCXPT=4
* NOTE: PRINT OUT INTRODUCTORY DIALOGUE FOR STAGE 4
* SELECT DIAL4
  #LLINE=1
  FREAD #LLINE,#S01
  DEPOSIT #S01\n  WRITE
  #LLINE=#LLINE+1
  IF #LLINE.LE.9,G0T0 1
* NOTE: SET DOMAINLIST AND DOMAINDICT RECORD POINTERS!
NOTE: CLEAR TOTALS INDICATOR
#GTOT=#0
NOTE: SELECT DOMAIN DICTIONARY, READ AND EXTRACT DOMAIN COUNT
SELECT DOMAIN
READ #LDDCT,#LDCT
CALLCONV(#LDCT,#LDCT)
NOTE: UPDATE DOMAIN POINTER, READ DOMAIN RECORD
#LDDPT=#LDDPT+1
READ #LDDPT,#SAI1
NOTE: EXTRACT FIELDS
VSTR2 #SAI1,,
READ #SAI1
#GINP=#GCCT+2
CALLVSTR(#LW4,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LW5,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LW6,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LW7,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LW8,0,,)
#GINP=#GINP+#GCCT+1
CALLVSTR(#LW9,0,,)
NOTE: PRINT DOMAIN NAME
DEPOSIT #SAI2
WRITE
DEPOSIT DOES THE ABOVE DOMAIN FEATURE IN YOUR PROBLEM? PLEASE TRY
CALLCOPY(#7,PESYFOR=YESFOR)
WRITE
DEPOSIT NO FOR NO. THE DEFAULT REPLIES IS NO.
WRITE
CALL 9444
SWIASH
IF #GSTC.GT.1,GOTO 4
CALLFSTR(#LW1,1,1)
IF #LW1.EQ.'Y',GOTO 5
IF #LW1.IN.'N',GOTO 16
GOTO 21MA,3
NOTE: BY DEFAULT INCLUDE KEY DOMAINS
IF #LW7.EQ.'N',GOTO 6
GOTO 7
NOTE: DOMAIN TO BE USED. TEST DOMAIN TYPE.
IF #LV5, EQ. 'A'
GOTO 7

NOTE: NUMERIC DOMAIN, ASK USER IF TOTALS REQUIRED
DEPOSITA DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL OF RETRIEVE

CALLCOPY(57, ORITDARY, THIS)
WRITE
DEPOSITA DO YOU WISH THE SYSTEM TO ACCUMULATE A TOTAL OF RETRIEVE

CALLCOPY(57, EPLYXIS%NO.,)
WRITE
CALL 9000
SQUASH
IF #GSTC, GT. 1, GOTO 9
CALLFSTR(#LV1, 1, 1)
IF #LV1, IN. 'N', GOTO 7
IF #LV1, EQ. 'Y', GOTO 14
OBEY 2100,

NOTE: SET TOTALS INDICATOR
#STOT=1

NOTE: SET TOTALS MARKER
#LV2='T'
GOTO 15

NOTE: CLEAR TOTALS MARKER
#LV2=' '

NOTE: SELECT DOMAIN LIST AND ENTER RECORD IN SIMFILE
SELECT DOMAINLIST
#LDLPT=#LDL^T+1
RESET #S91
DEPOSITA #SM2, #LV2, #LV4, #LV5, #LV7, #LV8, #LV9,
INSERT #LDLPT, #S91
SELECT DOMAINLIST
NOTE: REDUCE DOMAIN COUNT
#LDCT=#LDCT-1
IF #LDCT, NE. 0, GOTO 2

NOTE: UPDATE DOMAINLIST CONTROL RECORD
#LDLPT=#LDLPT-1
SELECT DOMAINDICT
REPLACE 1, #LDLPT

NOTE: CHAIN TO NEXT STAGE
[STAGES, STAGES]
GOTO 9999

NOTE: PRINT INVALID RESPONSE MESSAGE
DEPOSITA INVALID RESPONSE,
WRITE
RETURN

NOTE: ROUTINE TO READ USERS RESPONSE AND DETECT A PLEA FOR HELP

DEPOSITA :
WRITE
READ
CALLFSTR(#LV1, 1, 5)
IF #LV1, EQ. 'HELP', GOTO 9001
EXIT

CONTINUE
[CHAIN, HELP]
[HELP#1]

CONTINUE
THE FOLLOWING OPTIONS ARE AVAILABLE TO YOU IN RESPONSE TO YOUR PLEA FOR HELP:

1. ABANDON THE RUN.
2. RESTART THE DIALOGUE AT THE BEGINNING OF THE CURRENT STAGE OR AT A PREVIOUS STAGE.
3. TO SEE THE DIALOGUE FOR THE CORRESPONDING STAGE OF THE SAMPLE PROBLEM AND RESUME PROCESSING AT THE BEGINNING OF THE CURRENT STAGE.
4. TO EXPERIMENT WITH EDITING DATA AND THEN TO RESUME PROCESSING AT THE BEGINNING OF THE CURRENT STAGE.

AT WHICH OF THE ABOVE STAGE NOS. DO YOU WISH TO RESTART?

*DEF HELPY*  
*NOTE: HELP MACRO*  
*+ NOTE: EMPTY OUTPUT STACK OF ANY LEFT OVER RUBBISH*
%DEF HELP##
% NOTE: HELP MACRO
% NOTE: EMPTY OUTPUT STACK OF ANY LEFT OVER RUBBISH
% SAVE,0,WORK
% NOTE: PRINT INTRODUCTORY DIALOGUE
% SELECT DIALHELP
% NOTE: TEST CHECK POINT VALUE TO DETERMINE HOW MANY OF THE OPTIONS
% NOTE: SHOULD BE OFFERED TO THE USER
% IF #GCKPT.LT.9,GOTO 1
#LRANJ='123'
#LLAST=8
GOTO 2
%1
#LRANJ='1234'
#LLAST=10
%2
#LLINE=1
CALL 2000
% NOTE: REQUEST USER TO SPECIFY SELECTED OPTION
% DEPOSIT PLEASE TYPE THE NUMBER OF YOUR SELECTED OPTION, OTHERWISE
% CALL COPY(56,AULT%REPLY%1.)
WRITE
% NOTE: READ AND VALIDATE USERS RESPONSE
% CALL 9000
SQUASH
% IF #GSTC.GT.1,GOTO 4
CALL FSTR(#LV1,1,1)
% IF #LV1.EQ. ',GOTO 10
% IF #LV1.IN.#LRANJ,GOTO S
% NOTE: INVALID OPTION NO.
DEPOSIT0 INVALID OPTION#NO.
WRITE
GOTO 3
NOTE: SET DEFAULT RESPONSE
#LV1=1
GOTO 11
NOTE: BRANCH TO PROCESSING FOR SELECTED OPTION
CALLCONV(#LV1,#LV1)
#LV1=#LV1+5
GOTO #LV1
NOTE: OPTION 1 - ABANDON RUN
DEPOSIT0 RUN ABANDONED. PLEASE TYPE #FINISH
WRITE
EXIT
NOTE: OPTION 2 - RESTART AT AN EARLIER CHECK POINT
NOTE: COMPUTE LINE NO. FOR START OF STAGE LIST
#LLINE=29-#GCKPT
#LAST=29
NOTE: PRINT LIST OF AVAILABLE RESTART STAGES AND REQUEST
NOTE: RESTART STAGE NO.
CALL 2000
NOTE: READ AND VALIDATE STAGE NO.
CALL 9000
SQUASH
IF #GSTC.LT.1,GOTO 41
IF #GSTC.GT.2,GOTO 42
IF #GSTC.EQ.1,GOTO 42
CALLFSTRC(#LV1,2,1)
TESTDIGT(#LV1)
IF #GST,GT.4,GOTO 41
CALLFSTRC(#LV1,1,1)
TESTDIGT(#LV1)
IF #GSTC.EQ.4,GOTO 41
#Goup=#GSTC
CALLVSTRC(#LV1,1,0)
#Goup=1
CALLCONV(#LV1,#LV1)
IF #LV1.GT.#GCKPT,GOTO 41
IF #LV1.LT.#GCKPT,GOTO 41
NOTE: ADJUST CHECK POINT VARIABLE
#GCKPT=#LV1-1
NOTE: CHAIN TO REQUESTED STAGE
IF #GCKPT.GT.5,GOTO 3
CONTINUE
CHAIN#STAGE#LV1}
[STAGE#LV1#1=1]
GOTO 9999

NOTE: INVALID RESTART NO.
DEPOSITION INVALID RESTART STAGE NO.
WRITE
GOTO 43

NOTE: CHANGE MAINFILE

IF #GCKPT.EQ.13, GOTO 31
IF #GCKPT.EQ.14, GOTO 32
IF #GCKPT.EQ.15, GOTO 33
MAINFILE,GWAC132-DAT4
GOTO 45

MAINFILE,GWAC132-DAT1
GOTO 45

MAINFILE,GWAC132-DAT2
GOTO 45

MAINFILE,GWAC132-DAT3
GOTO 45

NOTE: OPTION 3 - PRINT SAMPLE DIALOGUE FOR CURRENT STAGE
NOTE: ESTABLISH RANGE OF LINES TO BE PRINTED

#LV1=1000+#GCKPT
GOTO #LV1

#1001 #LLAST=45
GOTO 1100

#1002 #LLAST=30
GOTO 1100

#1003 #LLAST=46
GOTO 1100

#1004 #LLAST=72
GOTO 1100

#1005 #LLAST=188
GOTO 1100

#1006 #LLAST=17
GOTO 1100

#1007 #LLAST=45
GOTO 1100

#1008 #LLAST=91
GOTO 1100

#1009 #LLAST=69
GOTO 1100

#1010 #LLAST=29
GOTO 1100

#1011 #LLAST=13
GOTO 1100

#1012 #LLAST=34
GOTO 1100
% GOTO 1100
%1014 #LLAST=40
% GOTO 1100
%1015 #LLAST=38
% GOTO 1100
%1016 #LLAST=24
% GOTO 1100
%1017 #LLAST=101
% GOTO 1100
%1018 #LLAST=221
%1100 #LLINE=1
% MAINFILE,GWAC132-ModL
% SELECT MODEL#GCKPT
% IF #GCKPT.EQ.17, GOTO 3000
% CALL 2000
%1101 MAINFILE,GWAC132-DATA
% #LV1=#GCKPT
% GOTO 44
% NOTE: OUTPUT MODEL17 SUBFILE
%3000 FREAD #LLINE,#S01
% NOTE: REPLACE > BY BLANK
% IF '<'.AT.#S01,GOTO 3001
%3002 DEPOSIT# #S01;
% WRITE
% #LLINE=#LLINE+1
% IF #LLINE.LE.#LLAST,GOTO 3000
% GOTO 1101
%3001 SWAP1,>
% GOTO 3002
% NOTE: OPTION 4 - TRIAL EDITING
% MAINFILE,GWAC132-DAT1
% [CHAIN],STAGE9
% [STAGE9@1]
% GOTO 9999
% NOTE: ROUTINE TO ISSUE PROMPT, READ RESPONSE AND DETECT HELP PFL
%9000 DEPOSIT# :
% WRITE
% CALLFSTRC#LV1,1,5)
% IF #LV1.EQ.'%HELP',GOTO 9001
% EXIT
% NOTE: PRINT REQUIRED LINES FROM SELECTED FILE
%2000 FREAD #LLINE,#S01
% DEPOSIT# #S01
% WRITE
% #LLINE=#LLINE+1
% IF #LLINE.LE.#LLAST,GOTO 2000
% EXIT
%9999 CONTINUE
%END