s-algol and the Commercial 3rd & 4th Generations

by

JOHN NORMAN SUTHERLAND, B.Sc.

Thesis submitted for the degree of

Master of Science

of the

University of St Andrews

October 1986

Abstract

s-algol and the Commercial 3rd & 4th Generations

This study compares s-algol with the commercial third and fourth generation languages and tools.

It divides into:

1) s-algol compared to COBOL and BASIC with reference to the relevant manuals with no actual coding.

2) recoding an existing BASIC system in s-algol which was abandoned due to lack of random access by key.

3) coding a new theatre bookings system in s-algol.

4) a fourth generation, interactive, code producer, written in s-algol, producing pseudo-ps-algol code was produced to write file maintenance programs. It was transcribed into COBOL and a batch data entry producer added. They were tested in parallel with a fourth generation language, VISTA.

5) A study of fourth generation tools was made and the opportunity shown to exist for algol in the fourth generation.

DECLARATION

I declare that the following thesis is a record of research work carried out by me, that the thesis is my own composition and that it has not been previously presented in application for a higher degree.

CERTIFICATE

I certify that John N. Sutherland
has satisfied the conditions of the
Ordinance and Regulations and is thus
qualified to submit the accompanying
thesis in application for the degree of
Master of Science.

A.J. COLE

DEDICATION

I dedicate this work to commercial computer programmers everywhere. As King Solomon said, "someone who holds back truth causes trouble, but one who openly criticizes works for Peace" (Proverbs 11.10).

ACKNOWLEDGEMENTS

My grateful thanks to Professor A.J. Cole for initiating this study, persevering with me and seeing it through to the end. Without his invaluable help and advice I never would have made it. To take a commercial analyst/programmer and produce an academic/commercial thesis on the direction and pros and cons of computer language design is a sign of the breadth of Professor Cole's horizons and the depth of his concern and understanding of Computing, its issues and problems. With more men of learning such as Professor Cole in universities and commerce there would be fewer misunderstandings between the users and designers of languages.

My grateful thanks also to my wife, Hazel. For seeing me through heights and depths. For listening to computer jargon. For allowing me out at night to write these programs.

My special thanks to the non-academic help of John Reavy, Mary-Claire McCabe, Karol Wojtyla and Joshua Judah.

My thanks also to John Kelly, Kath McPartland, Craig Dewar, Alex Scott, Michael Quine, Jon Watson, Jim Bone and Eilleen Dow for the help they provided in the Charles Letts (Scotland) Ltd. warehousing and Byre Theatre booking systems.

References - please note that for the purpose of this thesis references are implied using either

[m/n] or [m] or [m/n-p]

where

m = document number in the references section

n = start (or only) page number (inclusive)

p = end page number (inclusive) (except those publications where the pages are numbered a-b, where 'a' refers to the chapter and 'b' to the page within that chapter)

this covers documents with an author, editor or neither.

CONTENTS

Introduction

s-algol and its successor ps-algol are designed as genuine solutions to programming problems.

How do they compare with the standard commercial languages?

How do they apply to commerce?

How do they compare with the new fourth generation languages?

Have these algols any future in commercial data processing?

This thesis attempts to answer these questions in the following chapters by investigating what commercial languages are and whether the algols, for long the Cinderellas of programming languages, can seriously be considered commercial contenders.

Chapter 1

s-algol, COBOL and BASIC

1. Selection of Languages

This thesis considers s-algol as implemented on the DEC VAX under VMS.

Before any program design or coding could begin it was first necessary to find out what, if anything, the present s-algol lacks. It would be pointless to be in the process of coding a program only to find that the next command required was not present in the language. Also it would be incorrect and dishonest to give a language a stamp of approval knowing (or not) that it lacked the completeness required for commercial programming.

For these reasons the first task was that of evaluating what options are present in existing commercial languages that are not present in s-algol and deciding how useful and/or necessary these omissions are.

Many languages now used in commerce have arrived from various sources, by different routes and in various conditions. These languages should exhibit the facets which are necessary to enable the programmer to write for the business community.

There is no implication that these languages are perfect. But they should show the extensions that are absolutely necessary to add to s-algol to give it any extra facilities required without destroying its guiding tenet of "power through simplicity ... through generality" [22], as it is the lack of such a raison d'etre that has helped produce dialects of languages where the structure is obscured by the plethora of extensions.

The leading commercial language is COBOL [12/257-260]. This language is characterized by "file handling, manipulation of textual items, decimal arithmetic and very little numerical computational ability" [12/258]. COBOL will therefore be taken as a benchmark against which to measure s-algol.

However, despite the work of CODASYL (Conference on Data Systems Languages) to produce a standard COBOL their "standard" (my quotes) includes so many options for the compiler writer that there are several hundred similar versions. Each version relevant to the machine or machine range on which it is implemented. "Despite the attempts at standardization, variations from one computer's COBOL to another's still exist" [9/15] and "the manufacturer may select these features of the COBOL language which he considers most useful ... " [28/47]. This is especially true of some compilers which use only a small subset of the features of the language e.g. ICL 1900 COBOL.

To provide as complete a comparison as possible it is necessary to select a version of COBOL that uses as much of the CODASYL standard as can be found. I will at this point express a personal preference for IBM COBOL [14] as implemented on the System 360 range of machines as I have experience of coding commercial programs in this dialect. This may restrict the field so Honeywell COBOL [3] will also be selected for a comparison. Both versions use a great deal of the CODASYL standard.

This will not provide all the facilities necessary for a full comparison with s-algol as COBOL does not possess (by ANS (American National Standard)) any facilities for interactive processing, although there are aberrant strains such as Data General Interactive COBOL [16] which have imposed their own incomplete solution.

BASIC and APL are the leading languages in 'conversational' use [12/258-259] and the main language in use on microcomputers is BASIC. APL requires the user to have an extended and unusual character set on the terminal, printer and screen and has many unusual characteristics encouraging an "involuted style of programming" which is "complex ... esoteric ... terse ..." [12/258-259]. Not exactly s-algol's 'simplicity'. On balance I took BASIC as the better comparator.

BASIC dialects abound. Some manufacturers have more than one version of this "spaghetti style" programming language. I will here opt for Data General Business BASIC [6,7] (DGBB) as another language that will be used for the comparison. Once again this is a dialect that I have personal experience of using in commercial programming. It has powerful screen control facilities and is implemented on both micro and mini computers. As before, another manufacturer's dialect will expand the comparison. This is even more necessary with BASIC than with COBOL as it has no standard. A market leader in the mini computer market is DEC (Digital Equipment Corporation) and VAX-11 BASIC [34] (V11B) should provide a suitable fourth language for comparison purposes.

The following sections only extract the facilities available in COBOL and BASIC that are not present in s-algol and necessary for commercial programming.

2. s-algol in Relation to COBOL

error traps - through the USE verb fatal errors can be trapped within the program to allow an elegant exit. This is an option that is rarely used in COBOL due to its general batch usage. For on-line work it is almost essential as it allows an error to be caught without dropping the user into the command line interpreter (or equivalent). It also allows a neat message e.g. "File full - call programmer before continuing" as opposed to "ERR @f7, core dump follows...". This is an essential option for any serious on-line work.

data file accessing - ACCESS, ACTUAL KEY, RELATIVE KEY, NOMINAL KEY, RECORD KEY, OPEN, START, READ, WRITE, SEEK, REWRITE, CLOSE and DELETE when used in an indexed sequential processing environment have no equivalent in s-algol. Indexed sequential files are the core of commercial processing. Without an IFS (Indexed Filing System) - or other random access by key - commercial data processing would be almost impossible in any language, certainly more prone to errors from home brewed filing routines and slower to code. The ability to type in a code and get a specific record of details from a file relevant to the code or write details regardless of the uniqueness, size, content or index number (there may be more than one per file) with the same basic coding rules is, in a nutshell, exactly what commercial data processing is about.

3. s-algol in Relation to BASIC

error traps - BASIC provides an option which is probably the best part of the language: the error trap. ON ERR, RESUME and ON ERROR GO BACK allow an error to be trapped within the program; AERM$ and ERM$ provide the string arrays that hold the system error messages; SYS(2Ø) the line number of the source code of the last error; ERN$ the subroutine in which the last error occurred; SYS(7) the number of the last error. A system has a professional polish if errors are elegantly trapped, also providing security from the crashed program dumping the operator in the CLI where various dangerous commands may be typed.

Indexed sequential files - CLOSE, EOF, OPEN FILE, READ FILE, DELETE (record) and WRITE FILE are undefined in s-algol for IS (indexed sequential) files as are KFIND, KADD, KNEXT and KDEL. This is essential as mentioned for COBOL above.

character input - INPUT, INPUT FILE, INPUT USING and INPUT FILE USING provide the basic character input commands whether off a file or the vdu. (In practice used only for screen input.) (i) the ability to limit the maximum size of an input field is necessary to protect the screen layout and background. Neat i/o involves strict layout of fields so as not to overflow onto the background. S-algol should also provide this option (ii) the ability to position the cursor before an input should also be available (iii) there should be routines to handle all data types input which handle errors neatly within the program e.g. entering 'A' to a type real input statement should provide a neat error at the base of the screen, acknowledge by receiving a carriage return, clear the error message, re-input the value and also check whether the input is in the required range.

character output - PRINT, PRINT USING, PRINT FILE and PRINT FILE USING perform the character output functions for file and vdu processing in DGBB. Although there appear to be many screen control functions available they translate into a code or sequence of codes which are relevant to the VDU used. (i) VDU type handling should be transparent to the program (ii) s-algol should have access to as many vdu facilities as are relevant to commercial processing through standard calls (iii) the other options are akin to the formatting options provided by the COBOL PICTURE clause, allowing neat output of all data types and formats.

dates - STMA 11 and 12 provide a method of changing a date from day-month-year format to a julian integer value (total days from a base date). Apart from providing a simple format for comparing dates (standard DDMMYY format does not allow comparison of 'greater than' or 'less than' between dates) it also provides an excellent date validity test. Passing a date through the DDMMYY to integer conversion and back through the integer to DDMMYY conversion will validate a date e.g. 290283 will return as 310383. This is a must in s-algol for all date handling.

randomising - The pseudo randomiser RND and RANDOMISE are useful for spot checking of records in e.g. an audit. This option is popular with accountants and auditors and also those involved with stock control to check the computer record does reflect the real world. This would be a bit of icing on the cake if available in the DEC VAX VMS version of the language (it is present under CP/M).

4. Recommendations

In examining the COBOL's and BASIC's there is a warning to anyone wishing to add their favourite routines to a language. Firstly, the language soon becomes engulfed with a plethora of additions that may even duplicate part of another extension. These extensions also, paradoxically, restrict the power of the language as each extension often has restrictions on use which are then added to the existing restrictions. In use Business BASIC can only be used effectively, in my experience, by a programmer with at least six months experience of programming in the language. If not, there are many pitfalls in random disk accessing, system calls and others that will trip up the unsuspecting user. The language is complex allowing run time deletion of source code, alteration of specific memory locations, shutting down of the peripherals multiplexor and other horrors, some of which are actually necessary due to the design of the run time environment.

In addition there is the temptation to add a facility in a format that does not fit into the general language design. This makes a program ugly and difficult to use. An example of this is COBOL's usage of booleans by using level 88. Thus a part of the PROCEDURE DIVISION is relegated to the DATA DIVISION. There are many examples of this use of run time code being scattered around the various parts of a COBOL program. Another example of this is the COBOL verb, ALTER. It reallocates the destination of GO TO calls dynamically in the program run making the source code difficult to dry run and debug. In these I speak from experience of maintaining such programs in my full time post with the University of St. Andrews Administrative and Library Computer Unit (ALCU) and previous posts. Those programs with the ALTER verb are approached with trepidation.

These are the major caveats to be considered before making any recommendations: is the addition really necessary? how do I add it?

s-algol was not designed as a commercial programming language. Any commercial programming additions are not so much parts that the designers left out but ones that were not considered necessary in the production of a general purpose language that could be tuned to fit most computing tasks.

Thus the designer has the advantage of beginning with a compact and tidy language to which he can add his requirements. With carefully thought out additions the language should remain compact and tidy.

The gaps are nearly all i/o functions. This is not surprising when other algol variants e.g. Algol-W [11] are considered which are also deficient in the same i/o routines. Most can be coded as external procedures (see chapter 3) or as additions to the compiler options.

Specifically, the requirements for addition are:

(i) indexed sequential file processing or other random access by key.

(ii) function to input and output fields with type and range (for numeric fields) checking.

(iii) availability for character i/o (as relevant) of input field size control, vdu functions (cursor positioning, home, bell, new line, form feed, clear line/screen, brightness control, etc.).

(iv) error trapping.

(v) julian date conversion.

(vi) pseudo randomiser (but not essential).

In pointing out these deficiencies no evaluation has been performed on s-algol at this stage. Later (see chapter 3 section 5) there is an appraisal of s-algol after a complete system has been coded in the language.

Chapter 2

Charles Letts Warehousing System

1. System Selection

With s-algol viewed on paper in relation to the leading commercial languages and dialects the next step selected was to test the language in coding a practical commercial system. This would provide (i) a view from the programmer's angle of the language's facilities (ii) a view from the analyst's angle of the language's facilities.

Rather than tackle a new system in s-algol it was suggested that a system already implemented in another language should be used; perhaps one coded in one of the benchmark dialects. This would provide a more objective test of the language.

Any commercial computing personnel approached with a view to purchasing the language would want to know how it compares practically with the once ubiquitous COBOL and BASIC. To compare two identical systems would be a positive advantage.

BASIC, with its powerful screen control facilities is a broader language than COBOL in relation to the facilities available to the commercial programmer. My first approach was therefore for a system coded in BASIC.

The last system I implemented in Scotland for my previous employers, Fraser Williams (Scotland), a software house/consultancy, was in DGBB. This warehouse stock control system [40] would be suitable as (i) it was written recently (1981) (ii) it is coded in one of the previously used benchmark dialects (iii) my role was programming team leader, on-site implementation, technical author and user training (iv) it is a modern style menu driven system with no recourse to the CLI except to power up and down (v) it is currently

being used (vi) and, it works.

Since being employed by the University of St Andrews Administrative and Library Computer Unit (ALCU) the warehouse site systems controller, Mr Craig Dewar (Data Processing Manager, Charles Letts (Scotland) Ltd.) has kept in contact as regards the system for any operating system upgrades or queries. The staff of Fraser Williams have also kept in contact.

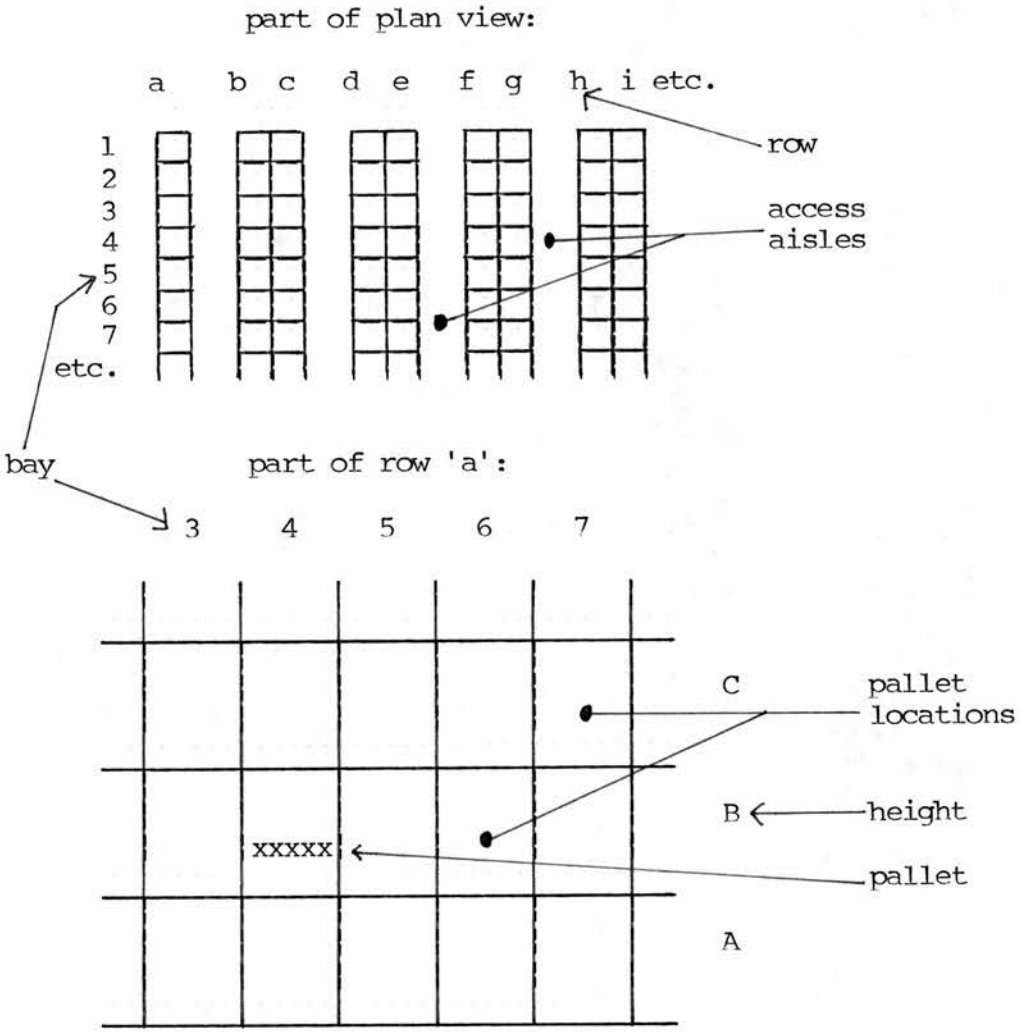
Accordingly I travelled to Glasgow to get the agreement of Fraser Williams (who hold exclusive distribution rights) and telephoned Charles Letts (who hold copyright). Mr D. Christie of the University's Quaestor and Factor's Department was also involved. Letters followed allowing the conversion along with a complete copy of the system documentation [18,19,20]

2. Problems

A problem was met very early on after the agreement on the recoding was reached. The principal file of the system is the stock/location file. This has two purposes (i) it identifies a pallet location in the warehouse (ii) it identifies the stock on a pallet within the location. However, for each location there may be more than one stock item. The pallet may hold, for example, glue and leather, if the containers are small. This is installed under DGBB using the language's duplicate key facility in its IFS.

As a second problem each item on a pallet has up to five other keys: stock, product, litho order, works order and/or purchase number.

figure 1.1 - Charles Letts Dalkeith warehouse



Thus a pallet could be located in row a, bay 4, height B.

As I made minor changes to the DGBB IFS routines during the initial DGBB coding to make them less prone to programmer error, I am aware of this problem and how it was tackled in DGBB. s-algol does not have the facilities available to the DGBB programmer. Thus facilities would be required to (i) access a location via its unique code (ii) access other records via the same location code (iii) access all records via other non-unique keys. As the facility exists in DGBB, then s-algol - a compact yet powerful language - could no doubt have such facilities added.

Despite my reluctance to tackle anything outside IFS, as by such I was straying far from my field of expertise, commercial data processing, three different tacks altogether were tried in attempting to get round this 6-indices problem within the scope of the direct file accessing facilities available in s-algol.

textbooks - these proved to be of little use, only describing what an IFS is and why use it [13/204, 13/271-280, 32/21-22, 32/30, 32/60-69].

software vendors - 105 compiler writing software vendors, as culled from the Computer User's Year Book [12] were mail shot to ascertain their help (see appendix A).

Only eight companies replied, perhaps due to the lucrative nature of the CP/M market at the time. Five could not provide help directly. One sent an application form for supply of electricity from the Irish Electricity Board!

OSW sent details of an ISAM package written in PASCAL and implemented under CP/M; but again no specific advice.

The last reply was from Beta Systems with, perhaps, a salutary comment "Binary tree (we did not think this worth implementing in our commercial programs after testing)".

manuals - finally, there is one place where commercial suppliers write down details of their products - in their manuals.

The DGBB IFS is a mix of separate index and data files. Thus, for the Charles Letts warehouse system there are physically six index files and one data file which together define the logical stock/location file. Index handling is primitive. There is the

standard binary tree structure with two problems (i) deleted index keys that leave an index block empty cannot have the block reused, thus the index must be tidied periodically (ii) block overflow causes a chain reaction along the blocks e.g. if a block has 4 keys present and 4 maximum then the addition of another key to this block causes the block to repack to two keys and the three remaining overflow onto the next block, and so on until the overflow stops; this also requires the resetting of upper levels of the tree to allow for the new block contents.

RMS, the IFS in V11B, is more complex than the DGBB IFS as (i) the indices and data are incorporated into one physical file (ii) the data file is in primary key order (iii) index blocks do not overflow sequentially. The consequence of (ii) and (iii) above is that overflow blocks are used extensively for data records and index entries.

In addition there are other complexities in RMS. Duplicate keys are not held physically as such in the index block and there is a facility to pack data, physically decreasing the file size.

3. Conclusions

At this point I was evaluating fourth generation computer languages for the ALCU central computer facility replacement and it was clear that COBOL was being dropped in favour of a better language. Rather than spend a considerable time applying an IFS to s-algol, the Charles Letts warehouse system implementation was discontinued on the grounds of lack of the necessary file i/o facilities.

Such facilities could certainly be coded into s-algol. But any IFS utilities added would be required to (i) work (ii) fit into the general language design. As a commercial analyst/programmer I could guarantee the latter but not the former in a reasonable amount of time.

Also, the s-algol compiler writers had produced a new variant, ps-algol, which does possess random access by key into a database. There was nothing to be proved by either pointing out again that s-algol lacked IFS when the compiler writers recognized this, or was there any point in adding IFS to s-algol when events had passed this necessity.

S-algol and ps-algol differ basically in the extensive and unique disk data storage facilities available in ps-algol. With a system selected that avoids the i/o problems that the warehouse system encountered then the implementation of such a system would also be a valid test for much of ps-algol.

Chapter 3

Byre Theatre Booking System

1. System Selection

The Byre Theatre, St. Andrews approached me through Mr. J. Bone to find a suitable booking system pre-written and available on the market. The full and proper course was followed with a market analysis. This aroused considerable interest from other theatres and bodies. The results were circulated around them: nine Scottish Theatres, Scottish Arts Council, Arts Council of Great Britain, City University Department of Arts Administration, City of Glasgow District Council Halls Department, Scottish Opera and the Calouste Gulbenkan Arts Foundation of Lisbon. With the conclusion that no system was available it was decided to attempt an s-algol implementation.

The system was capable, with a bit of juggling of files, of being coded in s-algol. A full system analysis was performed and produced followed by program specifications, external routines to fill identified commercial gaps in the language and finally the programs themselves.

2. Some External Procedures

Overview

I have attempted to produce routines that are easily used by the programmer, satisfy the user requirements and fit in with the general language design as well as other additions e.g. fformat, read.a.line.

The 42 added routines fall into five categories. They were bound into all programs as standard due to the considerable amount of common usage of code to avoid unnecessary overhead and code duplication:

(i) data processing

(ii) program sequencing

(iii) screen output

(iv) screen input

(v) file accessing

These are not all the possible externals that could have been coded into the system. There is a degree of duplicate code that showed up during program coding of similar programs which could be extracted and put in external routines. These were left in the programs to avoid confusion with the standard external routines which evidence the commercial additions to s-algol.

All programs were coded around a basic program skeleton which had all the necessary structures for file accessing, all the external procedure calls, and provided a standard format of program design for tidiness. The programs were compiled and bound by a command procedure to include all external procedures.

This chapter and the following one contain only examples of the work done. There follows only a few external procedures, the program skeleton and two complete programs with their associated documentation. A complete list and tape (where relevant) of external procedures, the program skeleton, programs, screens and outputs relevant to this chapter are all held at the Department of Computational Science at the University of St. Andrews, Scotland, as are the complete parts of the work referred to in the next chapter.

A considerable period of time was spent researching and reporting on the market for computerized box office systems and in later discussions with the Byre Theatre staff in analysing the existing manual system before specifying a proposed computer system.

With the system analysis completed the vdu routines were produced to allow all the required facilities. Then the filing system was designed. As can be seen below (see 'file access') this involved background work to grasp the use of the direct file type ,assessing how this file type is used in the RMS filing system and how it could be applied to the Byre Theatre.

Data Processing

The two procedures selected show how s-algol tackles two different problems: evaluating an integer and unpacking a string.

An integer is supplied to the following procedure that corresponds to a number of days from a base date, first of January, 1982 (produced by the routine julian.out - please refer to unbound appendix for this and other routines).

This routine outputs an integer that corresponds to the actual date referred to in the format DDMMYY. The two routines, julian.in and julian.out are used to (i) hold dates on disk in the number of days from a base date format (ii) compare dates without being limited by the non-arithmetic structure of standard UK DDMMYY usage (iii) verify dates - for example, pass 29/Ø2/85 through julian.out and then julian.in and the reply will be 1/Ø3/85; the inequality of input and output implies an invalid date.

```
!sk3
procedure julian.in(int julian->int)
begin
  external days.in.year(int->int)
  external days.in.month(int->int)
  let date:=84
  while julian>=days.in.year(date) do
  begin
    julian:=julian-days.in.year(date)
    date:=date+1
  end
  if julian=Ø then date:=date+3111Ø9 else
  begin
    date:=date rem 1ØØ + 1ØØ
    while julian>=days.in.month(date) do
    begin
      julian:=julian-days.in.month(date)
      date:=date+1ØØ
    end
    if julian=Ø then date:=date-1ØØ+days.in.month(date-1ØØ)*1ØØØØ
    else date:=date+julian*1ØØØØ
  end
  date
end
```

The following routine unpacks and verifies a string into a real number. It is used to allow all real numbers to be input as strings with data format errors being neatly handled within the program.

```
!sk5
procedure unpackr(string inp->real)
begin
  let outp:=Ø.Ø
  let zer:=48
  let negative:=1.Ø
  let point:=length(inp)
  if length(inp)>Ø do
  begin
    for i=1 to length(inp) do if outp~=maxreal do
    case(inp(i|1)) of
    ".":if point=length(inp) then point:=i else outp:=maxreal
    "-":if negative=1.Ø then negative:=-1. else outp:=maxreal
    default:if ~digit(inp(i|1)) then outp:=maxreal else
      outp:=outp*1Ø+decode(inp(i|1))-zer
    if outp~=maxreal do
    begin
      for j=1 to length(inp)-point do outp:=outp*Ø.1
      outp:=outp*negative
    end
  end
  outp
end
```

Screen Output

These procedures were specified for the Citoh Electronics CIT-1Ø1e [33] which emulates a DEC VT1Ø1. The codes are applicable on the DEC VT1ØØ [39].

The vdu lines are numbered 1-24 and the columns 1-8Ø. Line 1 carries the standard heading 'Byre Theatre ...'. Line 2 is blank. Line 3 carries the program and screen functions. Line 4 is left blank. Lines 5 to 21 inclusive are used by the program. Line 22 is left blank. Line 23 carries general input prompts. Line 24 carries error messages. (This is with the exception of the floor plan for booking as this requires 23 lines).

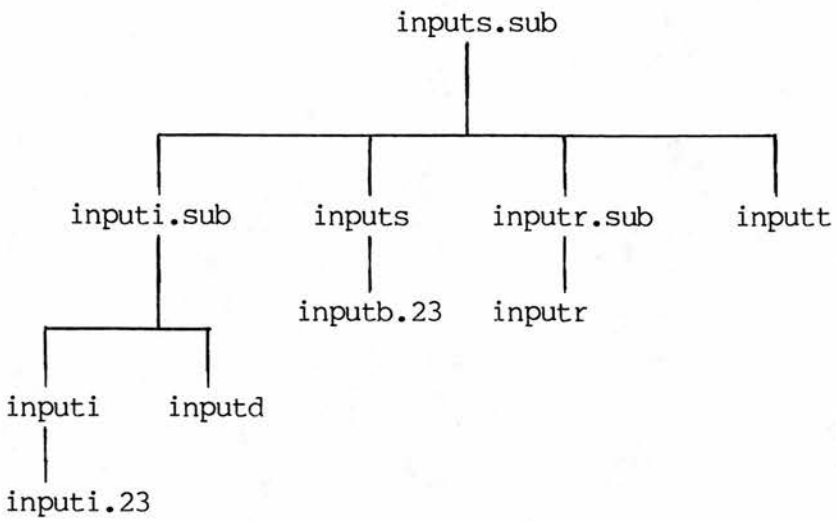
The following procedures show how s-algol handles a vdu transparently.

The first procedure below sends the standard code sequence to the vdu with the required option of the calling procedure. The second procedure performs various functions, for example, move the cursor relative to the present position, ring the vdu bell.

```
!skl4
procedure cursor.sub(string option)
begin
  let esc:=27
  write code(esc)++"["++option
end

!skl5
procedure cursor(string option)
begin
  external cursor.to(int,int)
  external fail(string)
  external cursor.sub(string)
  case option of
  "up":cursor.sub("A")
  "down":cursor.sub("B")
  "right":cursor.sub("C")
  "left":cursor.sub("D")
  "reverse":cursor.sub("7m")
  "blink":cursor.sub("5m")
  "uline":cursor.sub("4m")
  "bold":cursor.sub("1m")
  "reset":cursor.sub("Øm")
  "erase":cursor.sub("2K")
  "clear":{cursor.to(1,1);cursor.sub("ØJ")}
  "bell":write code(7)
  default:fail("invalid cursor option "++option)
end
```

Screen Input

structure of common calls:



The procedures all redisplay the data for neatness on screen and verification of input. e.g. input of "1" and "12.3Øam" will be both redisplayed by the procedure inputt with the same format " 1.ØØpm" and "12.3Øam".

All input routines, except inputb.23, will return a valid reply when no data is input (i.e. return only is hit). They will respond with the corresponding null value of the empty string or zero (integer or real equivalent) as relevant. This allows the operator to step through input routines using only the return key for defaults without having to constantly input values.

The selected procedure shows how a complex logical task appears neat and compact in s-algol.

The following procedure accepted and verified a time from the screen. It's complexity lies in that 12 noon is 12 p.m. and 12 midnight is 12 a.m., not vice versa as would be logically expected.

```
!sk30
procedure inputt(int row,column;real lower,upper->real)
begin
  external unpacki(string->int)
  external unpackr(string->real)
  external cursor.to(int,int)
  external printt(int,int,real)
  external inperr(int,int,int,string)
  external inputs.sub(int,int,int->string)
  let finished:=false
  let outp:=0.0
  while ~finished do
  begin
    finished:=true
    let inp:=inputs.sub(row,column,7)
    let fsize:=length(inp)
    if fsize>0 do
    if inp(fsize|1)="m" then if fsize=1 then finished:=false
    else
    begin
      outp:=unpackr(inp(1|fsize-2))
      if outp=maxreal then finished:=false else
      if outp<-0.001 or outp>12.001 then finished:=false else
      case inp(fsize-1|2) of
      "am":if outp>11.59 and outp<12.01 do outp:=0
      "pm":if outp<12 do outp:=outp+12
      default:finished:=false
    end
    else
    begin
      outp:=unpackr(inp)
      if outp=maxreal then finished:=false else if outp<12 do
        outp:=outp+12
    end
    if finished do if outp<-0.001 or outp>23.59 then
      finished:=false else
    begin
      let x:=fformat(outp,3,2)
      if unpacki(x(length(x)-1|2))>59 do finished :=false
    end
    if ~finished then
    inperr(row,column,7,
      "that''s not a valid time use e.g. 8.30am")
    else
    if fsize~=0 and (outp<lower or outp>upper+0.001) do
    begin
      finished:=false
      let low.err:=(if lower<13 then lower else lower-12)
      let low.am.pm:=(if lower<12 then "a" else "p")++"m"
      let up.err:=(if upper<13 then upper else upper-12)
      let up.am.pm:=(if upper<12 then "a" else "p")++"m"
      inperr(row,column,7,"please enter between"++
        fformat(low.err,3,2)++
```

```
      low.am.pm++" and"++fformat(up.err,3,2)++up.am.pm)
    end
  end
  if outp<0.01 then
  begin
    cursor.to(row,column)
    write "       "
  end
  else printt(row,column-1,outp)
  outp
end
```

File Access

In the absence of IFS another method of random file accessing was required to implement the Byre Theatre system. Relative files, which are alien to the commercial programmer, are the only building block available to s-algol.

On top of this was placed a simple file structure. The following procedures were designed and coded to allow precoded and tested routines to be used by programs to take the low-level i/o away from the coding.

The Byre Theatre consists of 145 fixed seats. The basic file could be designed around this fixed structure to allow a routine where the key was the seat number available transparently via a seek. This is where the Byre system differs from the majority of commercial systems which require similar random read/write but neither the number of the records nor their absolute file positions are known beforehand.

Only strings, integers and reals were used for i/o. Booleans wrote "true" and "false" which perhaps wasted disk space where a string "Y" or "N" (or "T" and "F" more logically with hindsight) would suffice.

To simplify the coding required to perform the reading and writing two decisions were made to limit complexity:

(i) All files were pre-set-up with empty fields

(ii) All writes had to preceded by a read.

These two combined to remove the necessity of creating new records or deleting records. Instead the records were flagged as used or unused. This also allowed the routines to verify each write command was valid by checking that it was preceded by a corresponding read.

The files designed were:

(i) System file - holds (a) system date (b) details of shows on the system.

(ii) Show file - one per show; holds (a) common show heading details (one per show file) (b) performance details (n per show) (c) seat details (145 per performance) (d) waiting list (1Ø per performance).

(iii) movements file - holds (a) number of last record used (first record) (b) first record number for a given date (set up with 2Ø dates, in practical terms would require many more) to allow fast access for a specific date (c) sequential records of bookings made (from record 22 onwards). This acts as a second index to the show files by date and time as it is a sequential record of all bookings made.

To open a file a procedure is called by:

```
let rec := set.file (rec.structure)
```

where rec is the pointer to the data ; rec.structure is the corresponding structure.

To read or write a file a procedure is called by:

```
access.file (key , io , rec , rec.structure)
```

Where key is the record identifier, io is one of "i" (for input) or "o" (for output), rec and rec.structure as above.

On top of the pseudo-IFS routines for open, read and write two more routines were added to read numbers off a file. Strings, Integers and reals are written in one operation. Strings are read in one operation. However, integers and reals require a second string dummy read to move the data pointer to the next available field. Two short procedures were coded to do this, freadi and freadr.

The other routines (i) produced a file name from a show name using the first (up to) nine valid characters (used for opening files) (ii) produced a performance key from a date and time (used for accessing show files)

The following routines show how structures and pointers are used by procedures as parameters allowing a flexibility not available in other commercial languages (see COBOL in chapter 4 section 5 below).

The following opened and accessed the system file

```
!sk37
procedure set.system(structure sy.str(int sy.d.date;string
  sy.s.name;*int sy.s.date;int sy.block;file sy.file)->pntr)
begin
  sy.str(0,"",vector 1::2 of 0,9999,open("system.dat","b",2))
end

!sk36
procedure access.system(string key,io;pntr sy.rec;structure
  sy.str(int
  sy.d.date;string sy.s.name;*int sy.s.date;int sy.block;
  file sy.file))
!
!key is one of:
!"date" - today's date
!"" or "<alpha>" - show record by name
!numeric - show record by number
!
begin
  external unpacki(string->int)
  external fail(string)
  external freadi(file->int)
  i.w:=1;r.w:=1;s.w:=0
  let f:=sy.rec(sy.file)
  let cr:="'n"
!access date record
  if key="date" then
  begin
    seek(f,0,1)
    case io of
    "o": if sy.rec(sy.block)~=0 then
    key:="not found" else
    output(f),sy.rec(sy.d.date),cr
    "i":
    begin
      sy.rec(sy.d.date):=freadi(f)
      sy.rec(sy.block):=0
    end
    default: key:="not found"
  end
!access show record by name
  else if length(key)=0 or ~digit(key(1|1)) then
  case io of
  "o": if sy.rec(sy.block)<1 or sy.rec(sy.block)>50 then
    key:="not found" else
  begin
    seek(f,sy.rec(sy.block),1)
    output(f),sy.rec(sy.s.name),cr
    for i=1 to 2 do output(f),sy.rec(sy.s.date)(i),cr
  end
  "i":
  begin
    seek(f,1,1)
    let reply:=0
    for i=1 to 50 do if reply=0 do
    begin
      sy.rec(sy.s.name):=read.a.line(f)
```

```
      for j=1 to 2 do sy.rec(sy.s.date)(j):=freadi(f)
      if sy.rec(sy.s.name)=key then reply:=i else
      if sy.rec(sy.s.name)="" then reply:=99
      else seek(f,1,0)
    end
    if reply>0 and reply<99 then sy.rec(sy.block):=reply
    else key:="not found"
  end
  default:key:="not found"
!read show record via record number
  else
  begin
    let key.i:=unpacki(key)
    sy.rec(sy.block):=key.i
    seek(f,sy.rec(sy.block),1)
    sy.rec(sy.s.name):=read.a.line(f)
    for i=1 to 2 do sy.rec(sy.s.date)(i):=freadi(f)
  end
end
```

3. Program Skeleton

All programs are built around this skeleton ensuring all external procedures are defined and called correctly and allowing the use of a standard command procedure for compiling and linking all programs.

```
!
!program:
!========
!
!function:
!
!
!structures
!==========
!
!system file
!
structure sy.str(int sy.d.date;string sy.s.name;*int sy.s.date;
  int sy.block;file sy.file)
!
!show file
!
structure sh.str(pntr sh.header,sh.performance,sh.seat,
  sh.seat.block,sh.waiting;int sh.sub,sh.block;string
  sh.key;file sh.file)
structure sh.h.str(string sh.h.name,sh.h.management;*real
  sh.h.prices;*int sh.h.date;string sh.h.dedreas;real
  sh.h.dedtot,sh.h.estimated,sh.h.actual;*string sh.h.terms)
structure sh.p.str(int sh.p.date;real sh.p.time;string
  sh.p.matinee;int sh.p.sold; *string sh.p.reports)
structure sh.s.str(int sh.s.seat,sh.s.date;string sh.s.staff;
  int sh.s.price;string sh.s.cancel,sh.s.cancelst,sh.s.comment)
structure sh.sb.str(*int sh.sb.seat,sh.sb.date;*string
  sh.sb.staff;*int sh.sb.price;*string sh.sb.cancel,
  sh.sb.cancelst,sh.sb.comment)
structure sh.w.str(int sh.w.entry;string sh.w.name,sh.w.phone,
  sh.w.required,sh.w.obtained,sh.w.comment;*string sh.w.address)
!
!all shows
!
structure as.str(*string as.name,as.filename;*int as.date)
!
!working storage
!
structure st.str( )
!
!
!procedures doing data processing and program sequencing
!=======================================================
!
external days.in.year (int->int)
external days.in.month(int->int)
external julian.in(int->int)
```

```
external unpacki(string->int)
external unpackr(string->real)
external julian.out(int->int)
external real.size(real->*int)
external day(int->string)
external chain (string)
external tformat(real->string)
external dformat(int->string)
!
!procedures to control screen output
!==================================
!
external cursor.to(int,int)
external clear.area(int,int,int)
external message(string)
external fail(string)
external cursor(string)
external error(string)
external display.screen(string)
external prints(int,int,string,int)
external printi(int,int,int,int)
external printr(int,int,real,int)
external printd(int,int,int)
external printt(int,int,real)
external inperr(int,int,int,string)
!
!procedures to control screen input
!=================================
!
external inputs(int,int,int->string)
external inputi(int,int,int,int->int)
external inputr.sub(int,int,real,real->real)
external inputr(int,int,real,real->real)
external inputt(int,int,real,real->real)
external inputd(int,int,int,int->int)
external inputi.23(string,int,int->int)
external inputb.23(string,string->bool)
!
!procedures to access files
!=========================
!
external freadi(file->int)
external access.system(string,string,pntr,structure (int,string,
  *int,int,file))
!where structure is sy
external set.system(structure (int,string,*int,int,file)->pntr)
!where structure is sy
external access.show(string,string,pntr,structure (pntr,pntr,pntr,
  pntr,pntr,int,int,string,file),structure (string,string,*real,
  *int,string,real,real,real,*string),structure(int,real,string,int,
  *string),structure(int,int,string,int,string,string,string),
  structure(*int,*int,*string,*int,*string,*string,*string),
  structure(int,string,string,string,string,string,*string))
!where structures are sh,sh.h,sh.p,sh.s,sh.sb,sh.w
external set.show(string,structure (pntr,pntr,pntr,pntr,pntr,int,
  int,string,file)->pntr)
!where structure is sh
external show.filename(string->string)
external show.key(int,real->string)
```

```
external get.all.shows(structure (*string ,*string,*int)->pntr)
!where structure is as
!
!
!procedures unique to
!=======================
!
!
!
!
!
!
!
!
!
!
!mainflow
!=======
!
?
```

4. Some Programs

Overview

The programs were designed with the user independent from the command line interpreter, DCL. The most common method I have used in the past, and one which is neat, is to use menus.

The user would log in using the standard account number and password. The login procedure would automatically call up the command procedure running the system. The first program is the one to set the date into the system file. Then the menu program is selected. Both are done automatically.

The user can now select the program he wishes to use. He selects by entering the relevant number (as per the menu screen) and pressing return. The program is run and after it is finished any further operations relating to the program are performed by the command procedure. These are (i) resetting the files after a new show has been created (ii) printing (in operation displaying is used for testing purposes) any output report or tickets (iii) ending the run gracefully if the program has crashed.

After the program and associated functions are completed the menu is redisplayed and a new selection can take place.

When the user wishes to finish using the system the logout option is selected. This sets the next program to the 'set date' program and logs out the user.

For brevity the programs have sections of repetitive skeleton code indicated by '...'.

As before, the following only represent a sample of the complete system. There are 17 programs in total performing menu control, ticket production, enquiry/update, reports and file set up. For the complete system there is an unbound appendix and tape available from the department which has the full requirement to set up the files and run the system on a DEC VAX under VMS with s-algol installed.

The two programs selected show how s-algol handles printouts, screens and files and the general structure of complete s-algol programs.

book a seat

name - bØ1Ø

resumé - to book seat(s) for a specific performance of a show, produce tickets and update the log and show files with the details.

procedure -

Display screen bØ1Øa. Open the system file and display show details name, date from and date to. Accept a show number. If the show number is Ø and there are no more shows chain to bØØØ; if there are more than one screen full of shows on the system file clear the screen and display the next batch of shows and return to input a show number again.

If a valid show number has been input open the relevant show file and display screen bØ1Øb. Display the performances' details of date, time, matinee and seats open (not sold). As for the show selection above input a performance number or display the next screen full of performances. If no performance was selected return to input a show again.

If a performance was selected display screen bØ1Øc. Do not use the standard procedure 'display.screen' as the whole screen is required. Display the seats in unpacked format (a1-z17) with open seats in reverse video and booked seats in normal video.

Display screen bØ1Ød. Allow input of a seat or seats. The format should be of the form 'x' or 'x-x' where 'x' is an unpacked seat number e.g. 'b5'. Check the combination is valid if a range is input. Check all the seats in the range are open. If not display an

error and reinput the seat selection. If no seats are selected return to select another performance.

Print the ticket into the file b010.pri.

Read the date off the system file. Update the show file: (i) write details to the seat(s) concerned (ii) update the number of seats sold on the performance record (iii) update the total value of seats sold on the header record.

Write a new bookings record to the file log.dat: (i) read the first record and rewrite with field value one higher (ii) position to the value read (iii) output a bookings record.

Chain to program p010.

source

```
!
!program: b010
!==========
!
!function:to book,cancel or wait on a seat
!
...
!
!working storage area
!
structure st.str(*int st.seats;real st.tot;string st.staff,
  st.comment;int st.price;pntr st.sh)
...
!
!procedures unique to b010
!=========================
!
!
!
procedure unpack.seat(string seat.s->int)
begin
  let out:=146
  case length(seat.s) of
  0:out:=0
  1:{}
  default:
  begin
    let ascii:=decode(seat.s(1¦1))
    let row:=(if seat.s(1¦1)>="a" and seat.s(1¦1)<="h"
    then ascii-decode("a") else
    if seat.s(1¦1)>="A" and seat.s(1¦1)<="H"
    then ascii-decode("A") else
    if seat.s(1¦1)="z" or seat.s(1¦1)="Z" then 8 else 99)
    if row < 99 do
    begin
      let column:=unpacki(seat.s(2¦length(seat.s)-1))
      if column>0 and column<(if row=8 then 18 else 17) do
      out:=row*16+column
    end
  end
  out
end
!
procedure format.seat(int seat->string)
begin
  let temp:=(if seat<145 then seat rem 16 else 17)
  code(decode("a")+(if seat<129 then(seat-1)div 16 else 25))++
  iformat(if temp = 0 then 16 else temp)
end
!
procedure format.seats(*int seats->string)
begin
  let start:=0
  let ends:=0
  let done:=false
  for i=1 to 145 do if ~done do
  case seats(i) of
  2:if start=0 then start:=i else ends:=i
```

```
  default:if start>0 do done:=true
  format.seat(start)++(if ends>0 then "-"++format.seat(ends) else
    "")
end
!
procedure select.show(pntr st.rec->bool)
begin
  display.screen("b010a")
  let sub:=0
  let sh.rec:=nil
  let as.rec:=get.all.shows(as.str)
  if as.rec(as.name)(1)="" then error("no shows on system") else
  begin
    for i=1 to 51 do if sub=0 do
    begin
      if i=17 or i=34 or i=51 do
      begin
        let top:=0
        for j=i-16 to i-1 do if top=0 do if as.rec(as.name)(j)=""
          do top:=j-1
        if top=0 do top:=i-1
        sub:=inputi.23("please select a show",1,top)
        if sub=0 do if as.rec(as.name)(i)="" then
        begin
          error("no more shows on system")
          sub:=51
        end
        else for i=5 to 22 do clear.area(i,1,68)
      end
      if as.rec(as.name)(i)~="" and sub=0 do
      begin
        let line:=i rem 17 + 4 +(if i>16 then 1 else 0)
        cursor("reverse")
        printi(line,1,i,1)
      cursor("reset")
        printd(line,4,as.rec(as.date)((i-1)*2+1))
        prints(line,13,"-",0)
        printd(line,15,as.rec(as.date)(i*2))
        prints(line,25,as.rec(as.name)(i),0)
      end
    end
  end
  if sub>0 and sub<51 do
  begin
    sh.rec:=set.show(as.rec(as.filename)(sub)++".dat",sh.str)
    if sh.rec(sh.file)=nullfile do fail("failed to open "++
    show.filename(as.rec(as.name)(sub))++".dat")
    access.show("h","i",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
      sh.sb.str,sh.w.str)
    st.rec(st.sh):=sh.rec
  end
  if sh.rec=nil then false else true
end
!
procedure select.performance(pntr st.rec->bool)
begin
  display.screen("b010b")
  let sh.rec:=st.rec(st.sh)
  let record:=0
```

```
  let opt:=0
  while opt=0 do
  begin
    record:=record+1
    access.show("p"++iformat(record),"i",sh.rec,sh.str,sh.h.str,
      sh.p.str,sh.s.str,sh.sb.str,sh.w.str)
    let sh.p.rec:=sh.rec(sh.performance)
    if (record rem 16 = 1 or sh.p.rec(sh.p.date)=99999) and
      record~=1 do
    begin
      opt:=inputi.23("please select a performance",1,record-1)
      if opt=0 do if sh.p.rec(sh.p.date)=99999 then
      begin
        error("no more performances in show")
        opt:=1000
      end
      else for i=7 to 22 do clear.area(i,1,45)
    end
    if opt=0 do
    begin
      let line:=record rem 16 + 6 +(if record rem 16=0 then 16
        else 0)
      cursor("reverse")
      printi(line,1,record,1)
      cursor("reset")
      printd(line,4,sh.p.rec(sh.p.date))
      prints(line,14,day(sh.p.rec(sh.p.date)),0)
      printt(line,24,sh.p.rec(sh.p.time))
      prints(line,33,(if sh.p.rec(sh.p.matinee)="y" then "matinee"
        else ""),0)
      printi(line,42,145-sh.p.rec(sh.p.sold),3)
    end
  end
  if opt<1000 do
  begin
    access.show("p"++iformat(opt),"i",sh.rec,sh.str,sh.h.str,
      sh.p.str,sh.s.str,sh.sb.str,sh.w.str)
    st.rec(st.sh):=sh.rec
    let seats:=st.rec(st.seats)
    for i=1 to 145 do
    begin
      access.show("s"++iformat(i),"i",sh.rec,sh.str,sh.h.str,
        sh.p.str,sh.s.str,sh.sb.str,sh.w.str)
      let sh.s.rec:=sh.rec(sh.seat)
      if sh.s.rec(sh.s.date)>0 do seats(i):=1
    end
    st.rec(st.seats):=seats
  end
  if opt=1000 then false else true
end
!
procedure display.stage(pntr st.rec)
begin
  let f:=open("b010c.vdu","s",0)
  if f=nullfile do fail("failed to open b010c.vdu")
  cursor("clear")
  let field:=""
  for i=1 to 4 do field:=read.a.line(f)
  prints(2,1,"on                    at",0)
```

```
  let row:=3
  while ~eof(f) do
  begin
    field:=read.a.line(f)
    if ~eof(f) do
    begin
      prints(row,1,field,0)
      row:=row+1
    end
  end
  close(f)
  prints(21,1,"row Z",0)
  cursor("uline")
  prints(21,41,"EXIT",0)
  cursor("reset")
  let seats:=st.rec(st.seats)
  let sh.rec:=st.rec(st.sh)
  let sh.h.rec:=sh.rec(sh.header)
  let sh.p.rec:=sh.rec(sh.performance)
  prints(1,1,sh.h.rec(sh.h.name),0)
  prints(2,4,day(sh.p.rec(sh.p.date)),0)
  printd(2,13,sh.p.rec(sh.p.date))
  printt(2,26,sh.p.rec(sh.p.time))
  for i=1 to 145 do
  begin
    let row:=((i-1) div 16)*2+5-(if i=145 then 2 else 0)
    let column:=case i<129 of
    true:(i rem 16 +(if i rem 16 = 0 then 16 else 0))*4+5
    default:case i<139 of
    true:(i-128)*3+6
    default:(i-138)*3+49
    if seats(i)=0 do cursor("reverse")
    let temp:=i rem 16 +(if i rem 16 = 0 then 16 else if i=145
      then 16 else 0)
    prints(row,column,(if i<129 then
      format.seat(i)(1¦(if temp<10 then 2 else 3)) else
      (if i<138 then " "else "")++iformat(temp)),0)
    if seats(i)=0 do cursor("reset")
  end
  message("press return when ready to select seats")
end
!
procedure select.seats(pntr st.rec->bool)
begin
  display.screen("b010d")
  let sh.rec:=st.rec(st.sh)
  let sh.h.rec:=sh.rec(sh.header)
  let sh.p.rec:=sh.rec(sh.performance)
  prints(5,1,sh.h.rec(sh.h.name),0)
  prints(6,4,day(sh.p.rec(sh.p.date)),0)
  printd(6,13,sh.p.rec(sh.p.date))
  printt(6,26,sh.p.rec(sh.p.time))
  for i=1 to 10 do
  begin
    let temp:=fformat(sh.h.rec(sh.h.prices)(i),3,2)
    prints(15,i*6+4,temp(2¦length(temp)-1),0)
  end
  let start.seat:=146
  let end.seat:=146
```

```
while start.seat=146 do
begin
  let start:=inputs(8,31,3)
  start.seat:=unpack.seat(start)
  case start.seat of
  0:{}
  146:error("please enter e.g. b17")
  default:while end.seat=146 do
  begin
    let ends:=inputs(8,38,3)
    end.seat:=unpack.seat(ends)
    case end.seat of
    146:error("please enter e.g. b17")
    0:
    begin
      end.seat:=start.seat
      clear.area(8,35,2)
    end
    default:if start.seat>end.seat do
    begin
      error("this must be on from the first seat")
      end.seat:=146
    end
  end
  if start.seat>0 and start.seat < 146 do
  begin
    let seats:=st.rec(st.seats)
    let ok:=true
    for i=start.seat to end.seat do if ok do if seats(i)>0 do
      ok:=false
    if ~ok then
    begin
      start.seat:=146
      end.seat:=146
      error("seat(s) already booked")
    end
    else
    begin
      st.rec(st.staff):=""
      while st.rec(st.staff)="" do
      begin
        st.rec(st.staff):=inputs(9,31,2)
        if st.rec(st.staff)="" do error("you must be someone!")
      end
      if sh.p.rec(sh.p.matinee)="y" then
      begin
        st.rec(st.price):=7
        printi(10,31,7,1)
        prints(10,34,"matinee",0)
      end
      else
      begin
        st.rec(st.price):=0
        while st.rec(st.price)=0 do
        begin
          st.rec(st.price):=inputi(10,31,1,10)
          if st.rec(st.price)=0 then
            error("you must select a price")
          else
```

```
            prints(10,34,(case st.rec(st.price) of
            1:"full"
            2:"concession"
            3:"student"
            4:"oap"
            5:"child"
            6:"quantity discount"
            7:"matinee"
            8:"unspecified 8"
            9:"unspecified 9"
            default:"unspecified 10"),0)
          end
        end
        let tot.seats:=end.seat-start.seat+1
        st.rec(st.tot):=tot.seats*
          sh.h.rec(sh.h.prices)(st.rec(st.price))
        let temp:=fformat(st.rec(st.tot),4,2)
        prints(11,31,temp(2|length(temp)-1),0)
        st.rec(st.comment):=inputs(12,31,15)
        if ~inputb.23("booking correct","y") do
        begin
          start.seat:=146
          end.seat:=146
          clear.area(8,31,3)
          prints(8,35,"to",0)
          clear.area(8,38,3)
          clear.area(9,31,2)
          clear.area(10,31,22)
          clear.area(11,31,7)
          clear.area(12,31,15)
        end
      end
    end
  end
  if start.seat>0 do
  begin
    let seats:=st.rec(st.seats)
    for i=start.seat to end.seat do seats(i):=2
    st.rec(st.seats):=seats
  end
  if start.seat>0 then true else false
end
!
procedure print.ticket(pntr st.rec)
begin
  let f:=open("b010.pri","a",1)
  if f=nullfile do fail("failed to open b010.pri")
  let sh.rec:=st.rec(st.sh)
  let seats:=st.rec(st.seats)
  let sh.p.rec:=sh.rec(sh.performance)
  let dmy:=julian.in(sh.p.rec(sh.p.date))
  let dd:=dmy div 10000
  let dd.s:=iformat(dd)++(case dd of
  1:"st"
  2:"nd"
  3:"rd"
  21:"st"
  22:"nd"
  23:"rd"
```

```
    31:"st"
    default:"th")
    let date:=dformat(sh.p.rec(sh.p.date))
    let time:=tformat(sh.p.rec(sh.p.time))
    let opt:=format.seats(seats)
    let tot:="        £"++fformat(st.rec(st.tot),4,2)
    output(f),"'pThe Byre Theatre, St. Andrews'n(0334) 76288'n'n"++
      day(sh.p.rec(sh.p.date))++" "++dd.s++" "++(case dmy div
        100 rem 100 of
      1:"January"
      2:"February"
      3:"March"
      4:"April"
      5:"May"
      6:"June"
      7:"July"
      8:"August"
      9:"September"
      10:"October"
      11:"November"
      default:"December")++", "++iformat(dmy rem 100 + 1900)++
      "'n'nat "++time++"'n'nseat"++(if length(opt)>3 then "s"
      else " ")++" "++opt++tot++
      "'n'n(tickets cannot be exchanged'nfor other performances"++
      " or'nmoney refunded)'n'n'n'n"++date++", "++time++
      "    STUB 1'n'n"++opt++tot++"'n'n'n'n'n'n"++date++", "++time++
      "    STUB 2'n'n"++opt++tot
    close(f)
end
!
procedure update.show.file(pntr st.rec)
begin
    let sh.rec:=st.rec(st.sh)
    let seats:=st.rec(st.seats)
    let tot.amount:=0.0
    let tot.seats:=0
    let sy.rec:=set.system(sy.str)
    if sy.rec(sy.file)=nullfile do fail("failed to open system.dat")
    access.system("date","i",sy.rec,sy.str)
    close(sy.rec(sy.file))
    access.show("h","i",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
      sh.sb.str,sh.w.str)
    let sh.h.rec:=sh.rec(sh.header)
    let sh.p.rec:=sh.rec(sh.performance)
    let perf.key:=show.key(sh.p.rec(sh.p.date),sh.p.rec(sh.p.time))
    for i=1 to 145 do if seats(i)=2 do
    begin
      access.show(perf.key,"i",sh.rec,sh.str,sh.h.str,sh.p.str,
        sh.s.str,sh.sb.str,sh.w.str)
      access.show("s"++iformat(i),"i",sh.rec,sh.str,sh.h.str,sh.p.str,
        sh.s.str,sh.sb.str,sh.w.str)
      let sh.s.rec:=sh.rec(sh.seat)
      sh.s.rec(sh.s.date):=sy.rec(sy.d.date)
      sh.s.rec(sh.s.staff):=st.rec(st.staff)
      sh.s.rec(sh.s.price):=st.rec(st.price)
      sh.s.rec(sh.s.comment):=st.rec(st.comment)
      tot.seats:=tot.seats+1
      tot.amount:=tot.amount+sh.h.rec(sh.h.prices)(st.rec(st.price))
      sh.rec(sh.seat):=sh.s.rec
```

```
    access.show("s"++iformat(i),"o",sh.rec,sh.str,sh.h.str,sh.p.str,
      sh.s.str,sh.sb.str,sh.w.str)
  end
  access.show("h","i",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
    sh.sb.str,sh.w.str)
  sh.h.rec:=sh.rec(sh.header)
  sh.h.rec(sh.h.actual):=sh.h.rec(sh.h.actual)+tot.amount
  sh.rec(sh.header):=sh.h.rec
  access.show("h","o",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
    sh.sb.str,sh.w.str)
  sh.p.rec:=sh.rec(sh.performance)
  access.show(perf.key,"i",sh.rec,
    sh.str,sh.h.str,sh.p.str,sh.s.str,sh.sb.str,sh.w.str)
  sh.p.rec:=sh.rec(sh.performance)
  sh.p.rec(sh.p.sold):=sh.p.rec(sh.p.sold)+tot.seats
  sh.rec(sh.performance):=sh.p.rec
  access.show(perf.key,"o",sh.rec,
    sh.str,sh.h.str,sh.p.str,sh.s.str,sh.sb.str,sh.w.str)
  close(sh.rec(sh.file))
end
!
procedure write.log(pntr st.rec)
begin
  let f:=open("log.dat","r",2)
  if f=nullfile do fail("failed to open log.dat")
  let cr="'n"
  let sh.rec:=st.rec(st.sh)
  let sh.h.rec:=sh.rec(sh.header)
  let sh.p.rec:=sh.rec(sh.performance)
  seek(f,0,1)
  let pointer:=freadi(f)+1
  seek(f,0,1)
  output(f),pointer,cr
  seek(f,pointer,1)
  let sy.rec:=set.system(sy.str)
  if sy.rec(sy.file)=nullfile do fail("failed to open system.dat")
  access.system("date","i",sy.rec,sy.str)
  output(f),sy.rec(sy.d.date),cr,sh.p.rec(sh.p.date),cr,
    sh.p.rec(sh.p.time),cr,show.filename(sh.h.rec(sh.h.name)),cr,
    format.seats(st.rec(st.seats)),cr
  close (f)
  close(sy.rec(sy.file))
end
!
!
!
!
!
!
!
!mainflow
!========
!
let st.rec:=st.str(vector 1::145 of 0,0.0,"","",0,nil)
let printit:=false
while select.show(st.rec) do
begin
  if select.performance(st.rec) do
  begin
```

```
    display.stage(st.rec)
    if select.seats(st.rec) do
    begin
      printit:=true
      print.ticket(st.rec)
      update.show.file(st.rec)
      write.log(st.rec)
    end
  end
  let sh.rec:=st.rec(st.sh)
  close (sh.rec(sh.file))
end
chain((if printit then "p010" else "b000"))
?
```

screens

screen A for program b010 (book seats): select a show

```
                       The Byre Theatre, St Andrews

Book a seat - please select a show

XXXXXX X XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

screen B for program b010 (book seats): select a performance

```
                       The Byre Theatre, St Andrews

Book a seat - please select a performance

   date      day      time          open

   XXXXXXXX  XXXXXXXX XXXXXXX  XXX   XXX
```

screen C for program b010 (book seats): stage

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
on XXXXXXXX XXXXXXXXX at XXXXXX
                                STAGE
                  ========================================


G                                                                      G
A                                                                      A
N                                                                      N
G                                                                      G
W                                                                      W
A                                                                      A
Y                                                                      Y



                 GANGWAY                              GANGWAY
row Z                                EXIT
```

```
screen D for program b010 (book seats): select a seats

                          The Byre Theatre, St Andrews

Book a seat - please select seats

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
on XXXXXXXX XXXXXXXXX at XXXXXX

   seat(s) required ..........XXX to XXX
   member of staff ...........XX
   price .....................X
   total cost ................XXXXXXX
   comment ...................XXXXXXXXXXXXXXX

Prices:
  1     2     3     4     5     6     7     8     9    10
XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX
 full  conc  stud   oap child  qnty  mat
```

printout

The Byre Theatre, St. Andrews
(0334) 76288

Friday 5th October, 1984

at 1.30pm

seats b7-b10 £ 8.00

(tickets cannot be exchanged
for other performances or
money refunded)

5/10/84, 1.30pm STUB 1

b7-b10 £ 8.00

5/10/84, 1.30pm STUB 2

b7-b10 £ 8.00

change or inquire on a show

name - bØ4Ø

resumé - This allows the user to inspect the details for a specific show. It allows you to inspect the details as input when the show was set up and gives details of how many seats are open (i.e. not sold) for the associated performances. The details can be amended for the show header record but not for the performance times. A performance may only be amended if no seats have been sold for this performance.

procedure -

Display screen bØ4Øc. Open the system file and display show details name, date from and date to. Accept a show number. If the show number is Ø and there are no more shows chain to bØØØ; if there are more than one screen full of shows on the system file clear the screen and display the next batch of shows and return to input a show number again.

Display screen bØ4Øa with relevant fields from the show header file. Allow amendment of management, deduction reason, deduction total, estimated income, and terms.

If requested display screen bØ4Øb and input a performance date. If the date is valid i.e. there are performances on that date then display these performances. A performance can be amended only if no seats have been sold. Allow amendment of the time and matinee flag only. Performances may not be deleted.

If required allow amendment of the show details again as above.

If amendments have been made then update them onto the show file.

Chain to b000.

source

```
!
!program: b040
!==========
!
!function:to amend or inquire on a show
!
...
!
!working storage area
!
structure st.str(string st.h.name,st.h.management;*real
  st.h.prices;*int st.h.date;string st.h.dedreas;real st.h.dedtot,
  st.h.estimated,st.h.actual;*string st.h.terms;*int st.p.date;
  *real st.p.time;*string st.p.matinee;*int st.p.sold;
  bool st.change)
!
...
!procedures unique to b040
!=========================
!
!
procedure display.options
begin
  cursor("reverse")
  for i=1 to 5 do
  begin
    cursor.to((case i of
    1:6
    2:9
    3:10
    4:10
    default:15),(case i of
    4:37
    default:2))
    write iformat(i)
  end
  cursor("reset")
end
!
!
!
procedure screen.b040a(pntr st.rec)
begin
  let field:=0
  display.screen("b040a")
  cursor.to(5,22);write st.rec(st.h.name)
  cursor.to(6,22);write st.rec(st.h.management)
  printd(7,22,st.rec(st.h.date)(1))
  printd(7,56,st.rec(st.h.date)(2))
  printr(9,22,st.rec(st.h.estimated),8)
  printr(9,56,st.rec(st.h.actual),8)
  printr(10,22,st.rec(st.h.dedtot),8)
  cursor.to(10,56);write st.rec(st.h.dedreas)
  for i=1 to 10 do
  begin
    let temp:=fformat(st.rec(st.h.prices)(i),3,2)
    prints(13,i*7+4,temp(2|length(temp)-1),0)
```

```
  end
  for i=1 to 6 do {cursor.to(i+15,2);write(st.rec(st.h.terms)(i))}
  display.options
  while field<6 do
  begin
    field:=inputi.23("please select field to amend",1,5)
    if field=0 then field:=6 else
    begin
      st.rec(st.change):=true
      case field of
      1: ! management
      begin
        clear.area(6,22,length(st.rec(st.h.management)))
        st.rec(st.h.management):=inputs(6,22,50)
      end
      2: ! estimated income
      begin
        clear.area(9,22,8)
        st.rec(st.h.estimated):=inputr(9,22,0.00,99999.99)
      end
      3: ! deduction amount
      begin
        clear.area(10,22,8)
        st.rec(st.h.dedtot):=inputr(10,22,0.00,99999.99)
      end
      4: ! deduction reason
      begin
        clear.area(10,56,length(st.rec(st.h.dedreas)))
        st.rec(st.h.dedreas):=inputs(10,56,15)
      end
      default: ! terms
      begin
        for i=16 to 21 do
          clear.area(i,2,length(st.rec(st.h.terms)(i-15)))
        for i=1 to 6 do st.rec(st.h.terms)(i):=inputs(i+15,2,78)
      end
    end
  end
end
!
!
!
procedure screen.b040b(pntr st.rec)
begin
  display.screen("b040b")
  let finished:=false
  while ~finished do
  begin
    let opt:=inputd(7,4,st.rec(st.h.date)(1),st.rec(st.h.date)(2))
    if opt=0 then finished:=true else
    begin
      printd(7,4,opt)
      prints(7,14,day(opt),0)
      let started:=0
      let ended:=0
      for i=1 to 50 do if ended=0 do
      if st.rec(st.p.date)(i)=opt then if started=0 do started:=i
      else if st.rec(st.p.date)(i)>opt do if ended=0 do ended:=i-1
      if started=0 then
```

```
      begin
        prints(7,26,"no performances",0)
        message("press return to continue")
        clear.area(7,4,40)
      end
      else
      begin ! print the current day's details
        let line:=7
        let amendable:=false
        for i=started to ended do
        begin
          printt(line,26,st.rec(st.p.time)(i))
          prints(line,38,if st.rec(st.p.matinee)(i)="y" then "yes"
            else "no",0)
          printi(line,47,st.rec(st.p.sold)(i),3)
          if st.rec(st.p.sold)(i)=0 do amendable:=true
          line:=line+1
        end
        if ~amendable then
        begin
          message("no performances can be amended")
          for i=7 to line-1 do clear.area(i,3,52)
        end
        else if ~inputb.23("change details","n") then
        for i=7 to line-1 do clear.area(i,3,52) else
        begin
          st.rec(st.change):=true
          printd(13,4,opt)
          prints(13,14,day(opt),0)
          let line:=13
          let last:=12
          for i=started to ended do if st.rec(st.p.sold)(i)>0 then
          begin
            printt(i-started+13,26,st.rec(st.p.time)(i))
            prints(i-started+13,35,if st.rec(st.p.matinee)(i)="y"
            then "yes" else "no",0)
          end
          else
          begin
            let ntime:=inputt(i-started+13,27,0,23.59)
            if ntime>0 do st.rec(st.p.time)(i):=ntime
            let nmat:=inputs(i-started+13,38,3)
            if nmat~="" do st.rec(st.p.matinee)(i):=(if
            length(nmat)=0 then "n" else if nmat(1|1)="y" or
            nmat(1|1)="Y" then "y" else "n")
            prints(i-started+13,38,if st.rec(st.p.matinee)(i)="y"
            then "yes" else "no ",0)
          end
          message ("press return to continue")
          for i=7 to 18 do clear.area(i,4,52)
        end
      end
    end
  end
end
!
!
!
procedure update.show.file(pntr st.rec)
```

```
begin
  let sh.rec:=set.show(show.filename(st.rec(st.h.name))++".dat",
    sh.str)
  if sh.rec(sh.file)=nullfile do
    fail("failed to open "++st.rec(st.h.name))
  access.show("h","i",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
    sh.sb.str,sh.w.str)
  let sh.h.rec:=sh.rec(sh.header)
  sh.h.rec(sh.h.management):=st.rec(st.h.management)
  sh.h.rec(sh.h.estimated):=st.rec(st.h.estimated)
  sh.h.rec(sh.h.dedreas):=st.rec(st.h.dedreas)
  sh.h.rec(sh.h.dedtot):=st.rec(st.h.dedtot)
  for i=1 to 6 do sh.h.rec(sh.h.terms)(i):=st.rec(st.h.terms)(i)
  sh.rec(sh.header):=sh.h.rec
  access.show("h","o",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
    sh.sb.str,sh.w.str)
  for i=1 to 100 do if st.rec(st.p.time)(i)>0.001 do
  begin
    access.show("p"++iformat(i),"i",sh.rec,sh.str,sh.h.str,sh.p.str,
      sh.s.str,sh.sb.str,sh.w.str)
    let sh.p.rec:=sh.rec(sh.performance)
    sh.p.rec(sh.p.time):=st.rec(st.p.time)(i)
    sh.p.rec(sh.p.matinee):=st.rec(st.p.matinee)(i)
    sh.rec(sh.performance):=sh.p.rec
    access.show("p"++iformat(i),"o",sh.rec,sh.str,sh.h.str,sh.p.str,
      sh.s.str,sh.sb.str,sh.w.str)
  end
  close(sh.rec(sh.file))
end
!
!
!
procedure select.show(->pntr)
begin
  display.screen("b040c")
  let sub:=0
  let st.rec:=st.str("","",vector 1::10 of 0.0,vector 1::2 of 0,"",
  0.0,0.0,0.0,vector 1::6 of "",vector 1::100 of 0,vector 1::100 of
  0.0,vector 1::100 of "",vector 1::100 of 0,false)
  let as.rec:=get.all.shows(as.str)
  if as.rec(as.name)(1)="" then error("no shows on system") else
  begin
    for i=1 to 50 do if sub=0 do
    begin
      if i=17 or i=34 do
      begin
        let top:=0
        for j=i-16 to i-1 do if top=0 do if as.rec(as.name)(j)=""
          do top:=j-1
        if top=0 do top:=i-1
        sub:=inputi.23("please select a show",1,top)
        if sub=0 do if as.rec(as.name)(i)="" then
        begin
          error("no more shows on system")
          sub:=51
        end
        else for i=5 to 22 do clear.area(i,1,68)
      end
      if as.rec(as.name)(i)~="" do
```

```
      begin
        let line:=i rem 17 + 4 +(if i>16 then 1 else 0)
        cursor("reverse")
        printi(line,1,i,1)
      cursor("reset")
        printd(line,4,as.rec(as.date)((i-1)*2+1))
        prints(line,13,"-",0)
        printd(line,15,as.rec(as.date)(i*2))
        prints(line,25,as.rec(as.name)(i),0)
      end
    end
  end
  if sub>0 and sub<51 do
  begin
    let sh.rec:=set.show(as.rec(as.filename)(sub)++".dat",sh.str)
    if sh.rec(sh.file)=nullfile do fail("failed to open "++
    show.filename(as.rec(as.name)(sub))++".dat")
    access.show("h","i",sh.rec,sh.str,sh.h.str,sh.p.str,sh.s.str,
      sh.sb.str,sh.w.str)
    let sh.h.rec:=sh.rec(sh.header)
    st.rec(st.h.name):=sh.h.rec(sh.h.name)
    st.rec(st.h.management):=sh.h.rec(sh.h.management)
    st.rec(st.h.estimated):=sh.h.rec(sh.h.estimated)
    st.rec(st.h.actual):=sh.h.rec(sh.h.actual)
    st.rec(st.h.dedreas):=sh.h.rec(sh.h.dedreas)
    st.rec(st.h.dedtot):=sh.h.rec(sh.h.dedtot)
    for i=1 to 10 do
      st.rec(st.h.prices)(i):=sh.h.rec(sh.h.prices)(i)
    for i=1 to 2 do
      st.rec(st.h.date)(i):=sh.h.rec(sh.h.date)(i)
    for i=1 to 6 do
      st.rec(st.h.terms)(i):=sh.h.rec(sh.h.terms)(i)
    let last.date:=st.rec(st.h.date)(1)
    let upper:=5
    let done:=false
    for i=1 to 50 do if ~done do
    begin
      access.show("p"++iformat(i),"i",sh.rec,sh.str,sh.h.str,
        sh.p.str,sh.s.str,sh.sb.str,sh.w.str)
      let sh.p.rec:=sh.rec(sh.performance)
      st.rec(st.p.date)(i):=sh.p.rec(sh.p.date)
      st.rec(st.p.time)(i):=sh.p.rec(sh.p.time)
      st.rec(st.p.matinee)(i):=sh.p.rec(sh.p.matinee)
      st.rec(st.p.sold)(i):=sh.p.rec(sh.p.sold)
      if st.rec(st.p.date)(i)=99999 do done:=true
    end
    close(sh.rec(sh.file))
  end
  st.rec
end
!
!
!
!
!
!
!
!
!
```

```
!mainflow
!========
!
!
!get show
!
let st.rec:=select.show
if st.rec(st.h.name)="" do chain("b000")
!
!amend details
!
let done:=false
while ~done do
begin
  screen.b040a(st.rec)
  if inputb.23("view performance details","y") do
    screen.b040b(st.rec)
  done:=~inputb.23("view show details","y")
end
!
!update files
!
if st.rec(st.change) then update.show.file(st.rec)
else message("no amendments made, press return to continue")
chain("b000")
?
```

screens

screen A for program b040 (view/amend show): view/amend header details

```
                        The Byre Theatre, St Andrews

View/amend a show -

Name.............XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Management.......XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Date from........XXXXXXXX          Date to..........XXXXXXXX

Estimated income.XXXXXXXX          Actual income....XXXXXXXX
Deduction amount.XXXXXXXX          Deduction reason.XXXXXXXXXXXXXXX

Prices:
full concess student  oap  child  qty  matinee  (8)   (9)   (10)
XXXXX XXXXX  XXXXX  XXXXX XXXXX  XXXXX XXXXX  XXXXX XXXXX XXXXX

   Terms:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

screen B for program b040 (view/amend show): view/amend performance details

```
                         The Byre Theatre, St Andrews

View/amend a show - please enter date

   date      day          times    matinee?  sold

   XXXXXXXX XXXXXXXX    XXXXXXX      XXX      XXX
```

screen C for program b040 (view/amend show): select a show

```
                         The Byre Theatre, St Andrews

View/amend a show - please select a show

XXXXXX X XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

7. Conclusions

To gain a true appreciation of s-algol it must be used. On paper it has few facilities. However, these provide all the requirements present in the morass of COBOL and BASIC, but in a neat, compact fashion.

The language has the full beauty of all algols, reduced to a minimum, with any extras providing real extra power (e.g. case and structure)

In the following paragraphs I have summarized the most noteable pros and cons from the commercial data processing viewpoint.

commercial or non-commercial? - when viewed from the commercial viewpoint s-algol comes across in use as a non-commercial language, similar to PASCAL in some respects. It is certainly not designed as a commercial language as COBOL is. Peek, readi, read, readr, readb, read, read.byte, sqrt, ln, sin, cos, atan, truncate, line.number, rabs, abs, eformat, gformat, shift.l, shift.r, b.and, b.or, epsilon, pi, r.w, flush and seek all exhibit this leaning towards low-level i/o and mathematical or some other non-commercial processing. This would be expected as the language was designed principally for this purpose, a design principle that does not prohibit its use for other forms of programming as the Byre Theatre suite evidences.

file structure - the lack of random access by key is the major limitation on s-algol in commercial use. The existing file access facilities together with recursive procedures could have such facilities coded. ps-algol does have random access by key into a database with more power than any commercial languages I have used or studied.

manuals - these are also aimed at a different market than commercial programming. With the simple addition of an index (as I made to the reference manual [22]) and commercial examples s-algol would be shown off to greater advantage.

real numbers - s-algol's real numbers are a different type of number from that used in, for example, COBOL. This is true of other languages not specifically designed for the commercial market. VAX-11 BASIC uses string arithmetic to avoid the rounding errors that scientific language real numbers - such as are used in s-algol - are prone to in large quantity calculations. Data General Business BASIC awkwardly sidesteps this problem by removing reals entirely from the language; the programmer is forced to use integers and 'remember' the actual number of decimal places.

This is not in itself a problem as string arithmetic procedures could be simply written in s-algol. The commercial programmer not brought up with real numbers as used in scientific languages would not, however, be au fait with the problems in testing for equality or the rarer problem of cumulative errors in adding very large quentities of real numbers which could be met in, for example, finance systems.

Thus for commercial processing, much of which is financial, string arithmetic procedures should be added before any further commercial systems are coded. But, this would not be a difficult task in s-algol.

constants - an interesting point is the use of constants in s-algol. Although not in the American National Standard COBOL [3/5-57], Honeywell and ICL provide this option while IBM, Data General, DEC and others do not. Thus constants are little used, although there is a school of thought that still backs their usage as

a separate data type. For example, most programmers have made the elementary mistake of building a print page maximum line number into a program instead of defining it at the top of the program only once. Changing the value for a subsequent different length page is more difficult if the line number is scattered around the program. s-algol does not force the programmer to use constants, but good programming practice should encourage it.

program structure - the design of the language is its strength as many errors that will pass a COBOL compiler or BASIC interpreter will not compile. Every variable must be initialized; every comparison must be of the correct type; every assignment of the correct type; every block must be complete; every procedure must be called properly; etc. One of the programs written for the bookings system ran correctly first time after it's first clean compilation.

Once compiled, any logic errors are more easily traced back to the code.

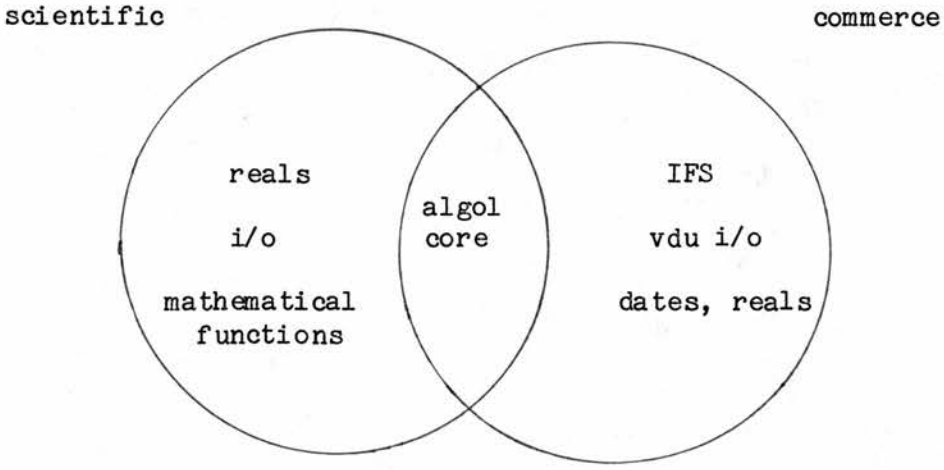
The most difficult part of debugging was getting used to errors often being flagged after the actual mistake, and with odd looking error messages ('type bool and void are incompatable in this context'!). But anyone using a COBOL reserved word in the wrong place in a COBOL program will be well aware of how easy it is to confuse a compiler when a piece of code is incorrect. What may therefore appear a disadvantage soon becomes apparent as an advantage as a clean compiled s-algol program contains signficantly less bugs than a correspondingly complex COBOL or BASIC program.

structures and pointers - the structure, structure class and pointers have a power that is quite amazing to the commercial programmer used to COBOL and BASIC. The ways in which data can be structured encourages the programmer to think of simpler, but at the same time, more powerful solutions to problems. They provide complete flexibility of core usage, allowing different data to be related and grouped together and cross-related with other groups of data. COBOL's usage of a maximum core map of fixed size is far excelled by this flexibility. Rather than explain it further on paper, perhaps the reader should look at the use made of structures, for example the show file structure, sh.str, in the programs above even though these only scratch the surface of this facility's power and flexibility.

These are not an exhaustive list of the differences between, say, COBOL and s-algol. They are what I found the most noticeable points of difference in comparison to existing commercial third generation languages, viewed as a whole. Given the lack of random access by key the language's small core of facilities all proved effective providing, in co-ordination with the externals coded, a commercial third generation programming environment.

final comments

It has been said already that commercial programming overlaps only partly with s-algol as provided:



and this is evidenced by programming in the language or reading the manuals.

Yet the language is a joy to use in many ways. Once the programmer grasps the structure of coding, he enters a different world from COBOL or BASIC. In s-algol the program is an entity, each block is an entity and each line also. In BASIC only the line counts, not its context. Thus, in s-algol the vast bulk of logic errors disappear. If the programmer writes a really bad program in COBOL or BASIC it will eventually compile and then prove to be a nightmare to run clean and thereafter to maintain. Give s-algol such a program and it will never even clean-compile.

But, s-algol is not prepared to enter commercial data processing. If s-algol (or any new language) had all the required facilities it would enter a harsh world. There are many commercial languages available, each with its own lobby who view it myopically and usually illogically as The Only language For Coding Anything In. I have met programmers who will defend RPG assiduously despite its resemblance to a football coupon. However, the recent experience of the ALCU (University of St Andrews, Administrative and Library Computer Unit)

has proved that where there is a will to escape from the confines of an old (or very old) language it can be done.

It should be of interest that the ALCU did not drop COBOL from new systems for reasons of language design but because COBOL could not handle screens and VISTA (a fourth generation language (4GL)) could - and much more. DEC COBOL struggles manfully with ACCEPT and DISPLAY statements for screen input and output; DG COBOL introduces a SCREEN SECTION that does allow partial scrolling, etc. But, there is no screen handling solution for COBOL yet.

As VISTA managed a foot in the door it became clear that it was a true 4GL and streets ahead in its facilities of any other tools then available on the DEC VAX under VMS. However, for data type definition, object code, compilation, linking, etc. s-algol is far superior; and that is not said as false praise for s-algol. Some of VISTA's anomalies are frightening e.g. (i) it often forgets what type and size a variable is (ii) source code anomalies - such as disallowing printing globally in a program but allowing it locally on a line causing the program to hang - are common.

The demand for a new type of commercial language is certainly here. But the limitations of COBOL are being exploited by completely different approaches to program design than the third generation languages.

The talk of the commercial world is the 4GL. These languages have arrived (but like COBOL, etc. are often in a pretty poor state of repair) and will continue to push out the old 3GL's.

This is perhaps the biggest problem for s-algol in its present form as a 3GL. The commercial market has powerful new tools that are more contemporary. Even with the above noted changes s-algol would be competing against more advanced tools. In the right circumstances s-algol could carve a niche for itself in the commercial third generation market. But, perhaps it could do better in the fourth generation market. This is the subject of the next chapter.

Chapter 4

The Fourth Generation

The following section, 'study of the existing market' has aroused interest in commercial circles as little independent work has been done in any depth on the fourth generation. Cognos Incorporated of Canada, manufacturers of the POWERHOUSE fourth generation suite, have requested and received a copy of this section of the study at their Livingston, West Lothian offices.

1. Study of the Existing Market

Early Attempts to Overtake the 3gl's

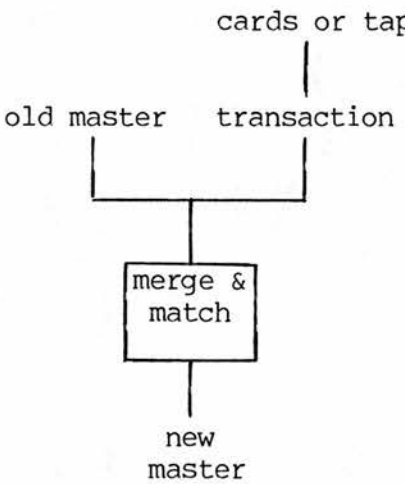
The idea of the 4GL has been present in commercial computing for quite some time.

Some earlier attempts were made, mostly scratching the surface, between the advent of COBOL (in many respects the commercial 3GL) and today's 4GL's.

The report generators were the most common. From fairly simple generators like ICL's FIND-2 package to the amazing RPG (Report Program Generator) [15]. Amazing only as to why it ever gained popularity.

Early commercial systems that the 3GL's and the early generators were written for were all batch systems. The vdu had not yet arrived. For this reason the most commonly met routines were (i) a master file update/transaction file merge and match:

figure 4.1 - batch merge and match



all file updating was of necessity of this type. (ii) report production.

Modern Problems

With the proliferation of vdu's the problems have changed and the 3GL's have notably failed to grasp this. COBOL has as yet no American National Standard screen control and subsequently some manufacturers have jumped the gun and added their own (e.g. Data General Interactive COBOL has a SCREEN SECTION [16/40ff]). BASIC varies with each compiler version. RPG's built-in program cycle cannot handle vdu's so extra, and again unique, routines are added to handle vdu's (IBM S/34 RPG2 has in effect a separate screen control program that handles all screen i/o and communicates with the actual program via screens of information at a time [15/12-1ff,15/13-1ff].)

The problem of printing is still a common one. This subdivides into standard reports (headings, detail lines, trailer lines) and statements (e.g. statement plus cheque stationery as used by the ALCU where a cheque statement may overflow onto separate pages [29/C1]).

File updating with data from outside the computer system still takes place in batch for a very few applications, notably inter-site data transfer, but data entry internal to an organization now takes place almost exclusively via the vdu. This subdivides into (i) master file updating e.g. adding new name and address details (ii) transaction processing e.g. adding items to a financial ledger.

To consider a real life system, the programs types written for finance work in 1984 at the ALCU were:

figure 4.2 - VISTA coding times

	coded in 1984	coding,testing & documenting days per program (%age of effort) 3GL	4GL
Data entry	6 (11%)	15 (29%)	8 (32%)
Document Update	12 (21%)	3 (11%)	1.5 (12%)
File Maintenance	15 (27%)	8 (38%)	2 (20%)
File print	12	2	Ø.5
Others	11	app 4	app 4
total		314	146

As can be seen the workload has diminished by over half (54%). Although this frees the programmer for other duties, screen handling has only reduced relatively from 67% to 52% of all tasks. There is an overall considerable drop but the programmer is still left relying heavily upon the screen handling facilities present in the language.

What is the structure of these repetitive programming tasks: file maintenance and batch data entry?

File Maintenance

Master file updating is usually through a file maintenance program. This allows insertion, amendment, viewing and deletion of one specific record. The key is entered and on (i) entry - data fields are entered, verified by the program, optionally amended and verified again; finally the record is either applied to the file or the insertion can be abandoned. (ii) amendment - the fields are displayed, amended, verified by the program and finally the changes are either applied or the amendment can be abandoned (iii) viewing - the details are displayed (iv) deletion - the details are displayed then they are either removed from the file or the deletion is

This type of program is also used for single record immediate updating onto a system. For example hotel room booking is of this type, while changing hotel room characteristics is file maintenance in the normal sense.

It can be seen that there are common routines. Figure 4.3 shows these routines and their relation to the four record handling options:

figure 4.3 - file maintenance cross-logic

	input	amend	show	delete
get option	X	X	X	X
read record		X	X	X
display fields		X	X	X
input fields	X	X		
verify fields	X	X		
update fields	X	X		X

the routines call each other as follows:

input calls amend
amend calls show
delete calls show

The result is that FM programs are very difficult to code, in particular as more fields are used. This is due to the complex nature of the screen handling as well as the tedium of coding 3GL repetitive routines with small differences.

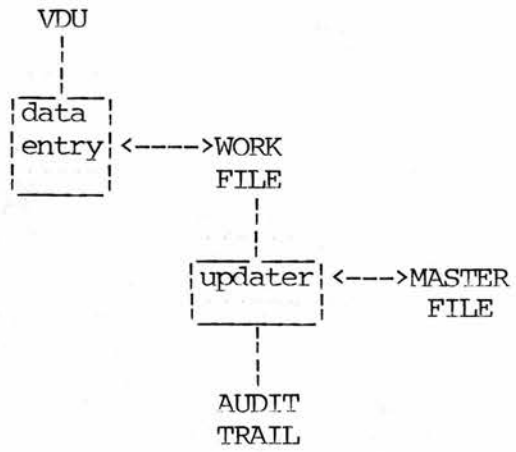
FM programs become more complex, almost uncodable, when multiple screens are involved.

Data Entry / Transaction Processing

The data entry suite is a variation on the old batch cycle:

The data entry suite is a variation on the old batch cycle:

figure 4.4 - data entry



The data entry program is probably the most complex program met in commerce. Like all screen programs it must be completely 'idiot-proof' to the data entry clerk(ess).

The program involves entering batches of data with header, detail and trailer options and optional control totals. Each record occupies 1 or 2 (or possibly more) vdu lines which scroll in the data entry area (usually lines 5 to 22). For example the University processes about 20,000 residence fees transactions every year [25]. The Cash Office divide these into batches for entry classified with common transaction type (e.g. invoice, credit note), term/quarter, date received. These details constitute the header record and each detail line is an individual transaction, e.g. a student paying his term 2 residence fee. At the end of a batch the computer checks whether the total value of the transactions add up and there are the correct number of lines in the batch. Batches can be viewed, amended or deleted also, as they must be if their totals do not balance.

The batches of data are applied to the database at a later time, usually overnight. Unlike FM, data entry is delayed updating of the type preferred on, for example, accounting systems.

The data entry program has all the facets of the file maintenance program with the additional problem of handling multiple records, multiple record types, variable screen positioning (the data entry scrolling area), control totalling and usually special function keys. The system of data entry is often referred to as transaction processing (an admittedly ambiguous name) and various tools have been available since the advent of the vdu to tackle this.

Whilst FM varies little from application to application and usually then at the whim of the programmer or analyst, TP is more flexible and complex yet also repetitive.

The Fourth Generation

It is with these specific problems and others in mind that the d.p. department turns to a 4GL. A considerable amount of time is spent in producing screen based programs (see figure 4.2 above) and printouts (4GL's are characteristically excellent screen handling tools). File maintenance, data entry, standard printouts and statement printouts are all faster to code, with fewer lines of source code and hence fewer bugs in the source code.

The 4GL compiler writers have taken further steps and added security features (logins, etc.), data dictionaries (see below for an example data dictionary) subroutine libraries and other facilities. It is here that problems often arise with the current 4GL's. To encompass all these a language/tool becomes rather complex. Some, like SYSTEL [4], are a real sledgehammer to crack a nut. There is so

much in them that the language/tool is unwieldy to use in programming and also expensive to purchase, although the user end-product is excellent. Systel has 19 manuals [4/3-3] and a variety of overlapping utilities and languages with cryptic names such as MLG, TCL, FRM, TCB and DDS. For example, the peripherals in a SYSTEL environment are required to be defined using the following format:

figure 4.5 - part of SYSTEL Configuration source file

```
LOCATION LOCNA
TERMINAL TTA1
DEVICE TTA1
TERMINAL TTA2
ON FEC ALPHA
DEVICE TTA2
PRINTER PRTA
DEVICE PRTA
TYPE MATRIX
LOCATION LOCNB
TERMINAL TTB1
DEVICE TTB1
AUTO ATTACH
LOCKED TRANSACTION JMSCON
WITH USERNAME FRED
AND PASSWORD 1
TERMINAL TTB2
DEVICE TTB2
AUTO ATTACH
TERMINAL TTB3
DEVICE TTB3
ON FEC ALPHA
TERMINAL TTB4
DEVICE TTB4
ON FEC BETA
TYPE VT52
PRINTER PRTB
DEVICE PRTB
TYPE MATRIX
```

The full file, held in SYSTEL.TBL, is processed by TBL producing SYSTEL.TCB and SYSTEL.PCB. These are subsequently used by TMC and DDS to control FEC and other devices [4/2-38,2-39]. Thus the programmer has a whole new area of complexity and jargon to contend with in using the tool.

Perhaps to the D.P. Manager the most obvious constraint on the 4GL is that its complexity often makes it environment dependant. The ALCU recently took COBOL from an English, non-ANSI compiler (3 versions, actually) to an ANSI compiler. This was relatively straightforward. VISTA would require a complete rewrite if removed from the DEC VMS or RSX environments. 'Cul-de-sac computing may appear an easy option in the short term, but the strategic importance of portability across supplier - and for that matter up and down a range of host systems - will sooner or later become apparent. In a world of mergers and takeovers ... be one step ahead by settling on a product that sees no bounds to either operating system or hardware.' [42/3]. This is borne out by the recent takeovers of two bidders for the ALCU's central computer replacement: ICL by STC and Systime by CDC (hardware) and DEC (maintenance), with the discontinuation of support for some months of the Systime 4GL product, Systel.

So the current 4GL's are a series of swings and roundabouts. I would never contemplate writing a screen or print program in a 3GL again as the 4GL does all the repetitive bits for you. However, the 4GL is one step further from the machine, as the 3GL was from the 2GL (assembler) and the 1GL (machine code). As there is so little source code the programmer is not always sure what the object code will do!

VISTA

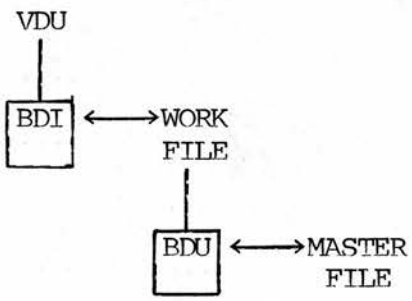
Chris Wimlett, a colleague in the University's Administrative and Library Computer Unit (ALCU), and myself surveyed the language market for the ALCU when the ALCU's central computing resource, the CTL 8Ø5Ø, was planned for replacement. We were advised particularly to view the DEC VAX VMS market.

VISTA [35,36,37] was chosen on price, simplicity and after visiting several reference sites. It was written for some printing and publishing houses in South East England and is now available through VISTA plc as a general tool on both PDP under RSX (the original environment) and VAX under VMS.

VISTA consists of 6 programs:

figure 4.6 - VISTA programs

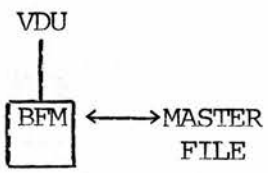
data entry, data update:



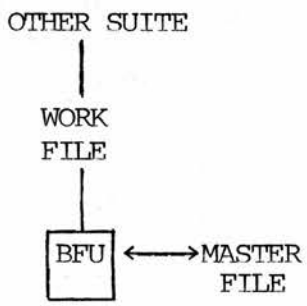
BDI = screen input

BDU = apply updates

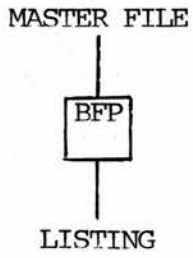
file maintenance:



file update:



file print:



statement print:

The various sites use different combinations of these. The ALCU uses BDI, BDU, BFM and BFP. It should be noted that BDU is only necessary as a separate option from BFU as the BDI work file is so appallingly designed.

VISTA also supplies a menu processor and tackles data dictionary and data security.

figure 4.7 - example VISTA menu

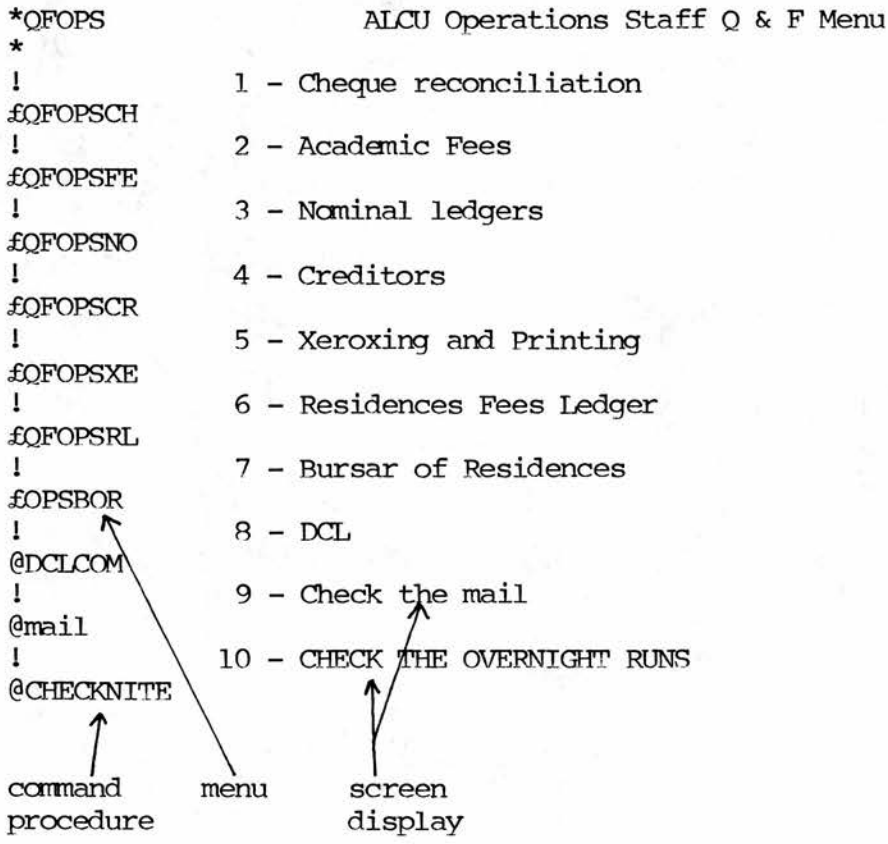


figure 4.8 - example VISTA file maintenance program

```
VISTA MAINTAIN "Xerox Codes", "XC", "XC.LOG", "XC.BAT", DISPLAY-STD=NO
*FMXE01
.FNAME          (T=30A)
.FNAM2          (T=30A)
LET   U.FNAME =         "SY5:[XEROX.DATA]XCODES.IDX"
LET   U.FNAM2 =         "SY5:[NOM.DATA]NOMACC.IDX"
FILE  XCODES   ^1,U.FNAME,"IM"
FILE  NOMACC   ^2,U.FNAM2,"IR"
INIT:
DISPLAY "Xeroxing and Printing Codes Maintenance"(@=P0000)
DISPLAY "U.St.A. Q & F"(@=P0066)
DOC-START:
DISPLAY         "File Code"(@=P0200)
INPUT ZK.MASTER(@=P0215,T=3C)
DISPLAY         ZK.MASTER(@=P0215,T=3C)
CLEAR XC.NAME
INSERT-DEFAULT:
CLEAR XC.RECORD
CLEAR XC.ACCOUNT(C=0)
LET   XC.FILECODE(T=3C) = ZK.MASTER(T=3C)
FIELD:
DISPLAY         XC.FILECODE(@=P0215)
CALL  SET-DELETE        XC.REC-STA
CALL  SET-MANDATORY     "0,1,2"
CALL  FIELD 00 XC.ACCOUNT       "Ledger a/c"            P0400   NOMACC
CALL  FIELD 01 XC.NAME          "Name"                  P0500
SUBROUTINE:   NOMACC
LET   ZK.MASTER(T=7A) =         XC.ACCOUNT
FILE-READ       NOMACC
IF    A
;CALL ERROR     "E>Not on accounts file"
LET   ZK.MASTER(T=3C) =         XC.FILECODE(T=3C)
```

Each program contains entry points. These are places where code is inserted by the programmer. For example, in file maintenance, there are entry points at (i) DOC-START - input record key from screen (ii) FIELD - handle i/o to record fields on the vdu.

VISTA has quite a few bugs. If a program fails it crashes the next one run through the processor and a VISTA program will lock out any other language program attempting to use any files used by the VISTA program, regardless of the mode opened. There are others.

Despite these, the simplicity in particular of FM and FP makes VISTA a valuable tool that leaves all existing 3GL's standing as regards speed of coding, maintenance and debugging of source code.

ps-algol

I have not tackled anything in ps-algol [5] on a live basis. However, the presence of a powerful random access by key into a database system fills the most obvious 3GL gap in s-algol.

ps-algol lacks none of the facets of the 3GL's originally considered (COBOL and BASIC). The question to be posed is however, how does ps-algol compare with the 4GL's, which have only arrived recently on the d.p. scene.

ps-algol is a 3GL. It does not possess the prewritten routines for screen handling, printouts, etc. Neither does it have the security features and higher level facilities that make the 4GL's so environment specific and often unwieldy to use.

Subroutine libraries as used by the 4GL's are also present in ps-algol through its data base handling. Ps-algol's consistent approach to storing information does not show up in the approach of the 4GL writers to data, routines and data dictionaries; they are not held consistently with respect to each other.

The 4GL's use of data dictionaries and auto-documentation are powerful facilities which ps-algol lacks.

Auto-documentation covers up to three areas: (i) system documentation - this aids the analyst in defining files/database structures and programs (ii) program documentation - a programming aid used to define programs to aid maintenance (iii) operating

documentation - user aid to define how the program works and what to use it for. Documentation is only produced by suites with a considerable amount of central logic predefined. Thus VISTA does not produce any program or operating documentation as much is left to the whim of the analyst, but VISTA could provide system documentation if necessary. A true 3GL is too flexible to provide any auto-documentation, except possibly system documentation if it uses a database.

A data dictionary is very much a fourth generation concept. No field may be accessed from a database from within any program unless that field is defined on the data dictionary by name. The data dictionary entries define files, records and fields. In particular fields are defined into type, size, name, range, values, etc. At field level the principal advantage is that a field is only so defined once and can then be used in many programs and files in the knowledge that the suite is completely consistent in its use of database fields. Programs in turn extract from the data dictionary the relevant details of fields required; a data entry program would require all details, while a print program would only require size and type. At a more powerful level (e.g. PRO-IV) the data dictionary can be altered and all programs' references to this field are correspondingly adjusted. Ps-algol or s-algol could have field verification routines written into external procedures and bound in at run time. In addition it may be possible to include other data dictionary concepts into ps-algol but, as will be seen later, this would be akin to adding IFS to s-algol (see chapter 2 section 3 above), quite unnecessary as a better solution is available. As ps-algol has a considerably more powerful database accessing and storage method so the design philosophy of the fourth generation will be seen to be applicable to ps-algol,

encompassing comfortably the data dictionary concept. I would suggest leaving the data dictionary to a fourth generation ps-algol (see example of a data dictionary and the consideration of ps-algol in the fourth generation below in this section).

The 4GL's also have commands that cannot be expanded to a simple routine. FIELD [35/43-44], as used by VISTA, defines where a record field is to appear on the screen. From this command comes the code to input, verify, display, etc. this field. But this also interacts with the code for handling other FIELD commands. Therefore, VISTA sorts the 3GL source code produced by it and puts it into the order it prefers; hence why I said above that the programmer is not always quite sure what the object program is doing; the compiler has a modicum of intelligence built in.

figure 4.9 - VISTA FIELD call

```
CALL     FIELD 01  XC.NAME   "Name"    P0500  NOMACC
{   a       } {b} {  c  }   {  d  }  { e }  {  f }
```

a - command

b - number of field on screen if amending is required

c - file identifier and field name together defining the field in the program

d - literal to be displayed on the screen

e - absolute position of field; row 5, column zero (numbering both row and column from zero). This is the position at which the 'b' is displayed. 'd' is displayed four columns on. Then the field is input 16 characters on from the initial given position.

f - name of subroutine to perform input verification checking.

As with s-algol, a ps-algol file maintenance program would require to be painstakingly written over a considerably longer period of time than a 4GL FM program. The algol structure will, however, ensure that the object program will be far freer from bugs than an equivalent in the generally used on-line language, BASIC. This is also true in relation to the true 4GL's to an extent as the source code provided by the programmer is written in a 3GL-like language with GOTO's that go nowhere, uninitialized variables, and all the concomitant problems carried over from the old 3GL's.

Conversational Tools

It therefore follows that the true 4GL's are ahead of ps-algol in speed of coding. But the 4GL market does not only consist of new languages. What ps-algol could do is tackle the 4GL's by going one better. This may appear presumptuous as it took the 4GL's 20 years to appear after the 3GL's. However, the aim of the 4GL is to increase programmer productivity, not of necessity to produce a new language, after all, the language is only a means to an end. That end being the removal of actual coding.

With this in mind I wrote a file maintenance program generator (see section 2 below in this chapter). This interacted by asking file and fields details. For each field it asked name, whether it was a key, type, occurrences, display name, valid values, valid range of values and maximum size. The interactive session was very simple and the actual code produced may not have been bug-free ps-algol and certainly was not good ps-algol but that was not the point being made. The aim was to prove and produce a program that would produce a file maintenance program (already noted as being a difficult and recurrent

commercial program) that worked, was neat, well documented and logically structured, in no more than a few minutes. Normally one of these programs would take several days to write in s-algol (as they did for the Byre Theatre system (see bØ4Ø in chapter 3.4 above)) and the resulting code would not be as exact and neat as produced by the fm generator. Also the humanly produced program could have hidden bugs; once such a fm generator was proved it would produce exact working programs every time. In addition the generated program could produce an audit of all actions performed by the user and could handle multiple screens of information.

The tool certainly worked in that it produced a logically correct program from a short question and answer session as it was designed to do.

Similar tools could be written to handle data entry in particular and with further design the option to produce standard print producers and statement producers could also be tackled. All are reducable to a body of logical repetitive sections from which a good generator can produce a working program consistently.

The proof of the pudding is certainly in the eating in commercial data processing. With this in mind I made the decision to move the tool from possibly ps-algol under UNIX to ANSI COBOL under VMS. In addition the tool was implemented on the University's Administrative and Library Computer Unit DEC VAX 751.

In use it soon became obvious that the central core of logic was correct but the tool (given the name, FMG (file maintenance generator)) was too primitive. The tool was further enhanced as

(i) the question and answer session was divided from the actual generator, with the information stored on disk.

(ii) the generator was then used to produce an interactive version of itself.

(iii) the documentation generator was added

(iv) the common routines were extracted into COBOL subroutines and bound into a library in the same manner as the Byre Theatre project s-algol routines were held and compiled separately as external routines (see chapter 3 above).

In addition a transaction processing generator (named TPG (transaction processing generator)) was produced. This emulates the VISTA BDI and BDU programs (see figures 4.4 and 4.6 above) in that it handles multiple record types and batch totalling. It produces two programs with the relevant operating documentation.

Finally a standard print program generator was added. (Incomplete at this stage and certainly not the final possible addition).

There arose a consistent and logical method for producing a generator: (i) write a very neat program that performs everything you would require at a maximum (e.g. in fm multiple screens, multiple keys) and ensure it works (ii) extract and compile all possible repetitive subroutines (e.g. input a string, position cursor) (iii) compile and prove program with these routines as external subroutines (iv) extract other static pieces of code (e.g. COBOL DIVISION headers) into a separate file as a skeleton program with code insertion points for the remaining variable codings (v) identify the minimum parameters required to produce the variable codings. (vi)

write and prove the generator to produce these variable codings in co-ordination with the static details held separately, outputting the fm program (vii) in the case of fm use the generator to produce an fm program to input the parameters (see (v) above); this program is then used to front the generator.

This method of reduction was used to produce the COBOL generators as it was not fully identified until after the algol generator was completed. Full time equivalent the algol generator took about four weeks to produce, the COBOL fm generator about 2 months and the COBOL tp generator about two months also. Once again, the ease of coding in s-algol is apparent. The difficulties met in returning to programming in COBOL (despite the fact that this is the principal language I use in my full time position) were apparent as logic bugs reappeared, convoluted codings were added (for example the ADD-FIELD SECTION in FMTWO.COB), etc.

The above method could also be used to reduce other repetitive data processing program types to a simple generator using the minimum parameters, for example, statement production (forms with header, detail and trailer areas with possible multiple page overflow).

The COBOL utilities were used to produce programs for the University's Works Department system (which is still under development) and to replace the VISTA BDI and BDU (transaction processing) programs within the nominal, residences and creditors ledgers posting systems which do not fully work as required [2/16ff,2/42ff,8/2ff]. This produced the following interesting chart of speed of use:

figure 4.10 - FMG/TPG coding times

	coded	coding,testing & documenting days per program->total time 3GL	4GL	FMG/TPG
Data entry	3	15->45	8->24	1->3
Document Update	12	3->36	1.5->18	1.5->18
File Maintenance	17	8->136	2->34	1->17
total		217	76	38

Again the increase in productivity of the 4GL over the 3GL is noted in screen handling program production. But the speed of FMG/TPG over both, an increase in productivity of a factor of 5.7 over the 3GL and 2 over VISTA was unexpected. In particular the problems previously encountered using VISTA's data entry suite are evidenced by the increase in productivity of 15 over the 3GL and 8 of TPG over BDI.

Ongoing maintenance is the second consideration when programming. On the CTL 8050 minicomputer that preceded the A.L.C.U.'s VAX 751 until 1984, all transaction processing (meaning data entry, data enquiry and file maintenance) was via the language/suite TAD [21]. No further maintenance was possible after one of the Unit's staff left. The TAD system required complete rewriting in VISTA BDI, BDU, BFM and BFP when the systems transferred to the VAX. However, the accounts data entry system requires batch control totalling not available in BDI. Attempting to code this option in has proved only partly successful, consequently the BDI programs have never worked completely. This evidences further problems that can often be encountered. No matter how rapidly a system can be developed using a tool (i) in time a non-standard coding will be required that may prove impractical (ii) the tool's complexity could make it difficult to maintain or (iii) changing machine may prove difficult. Hence why FMG

and TPG have replaced VISTA for the latest ALCU system.

The fm generator in particular could be further enhanced with major alterations to allow:

(i) cross-file checking of fields allowing data validation, data description display and most usefully would inhibit record deletion of related data items.

(ii) field subdivision to allow entry of a field that subdivides into different entities for checking purposes.

(iii) data dictionary of field commonality

(iv) dynamic field input, only enter field A if field B contains C.

(v) multiple programs per file.

Such a generator falls directly into the sharp end of commercial language/tools design.

Existing Tools

Since 1983, when the ALCU selected VISTA, there has been a major expansion of the commercial tools market. This can lead to confusion as products are often marketed under unsuitable labels with extravagant claims.

NOMAD/2 claims, "If the package required ... the needs of a dp professional, the overall objectives will not be attained" [24]. Yet, " ... don't believe the standard myth that 4GL's are true end user tools. In fact, fourth generation languages can only be used as end-user aids in simple, non-integrated systems that are relatively small" (Whiteside, [42]). Whiteside is correct as the NOMAD2 writer

has forgotten that the 4GL tools are aimed at replacing 3GL programming, not the related but different and far more complex discipline of systems analysis.

The question often asked is 'what is a 4GL?'. Consequently marketers rubbish the opposition and promote their particular solution. I would turn it around to 'why is the 4GL?'. The market is for tools that increase and enhance programmer productivity. The market would perhaps be better described as 4th generation tools as the concept of the language has often been discarded completely.

To give an overview, but no more than that, of the currently available products I mailshot a number of identified marketers of such tools (see appendix C). A good number responded allowing the following view of the central themes to these tools:

(i) new language - VISTA, ADR, SPEEDWARE, INFO, POWERHOUSE, ADS/ONLINE, SYSTEL, NOMAD/2, NATURAL.

(ii) database (relational or otherwise) - ADR, USER-11, ADS/ONLINE, NOMAD/2, ADABAS, PRO-IV, MICS, FOCUS, ULTRA, SPEEDWARE, INFO, MIMER, INFORMATION, DATAFLEX.

(iii) data dictionary - ADR, USER-11, ADS/ONLINE, SYSTEL, NOMAD/2, PREDICT, PRO-IV, MICS, ULTRA, INFO, POWERHOUSE, INFORMATION.

(iv) auto-documentation - ADR, ADS/ONLINE, NATURAL/ADABAS/PREDICT, DELTA, PRO-IV, MICS, SPEEDWARE.

(v) run-time help - ADR, ADS/ONLINE, PRO-IV.

(vi) 3GL code producer - DELTA (COBOL and PL/1), USER-11 (BASIC+2).

(vii) database query languages - MIMER, INFORMATION, DATAMASTER, ULTRA, DATAFLEX.

(viii) collection of tools - FOCUS/FIDEL, DATAFLEX

The variability of the products is thus immense and many claim an impressive list of clients: VISTA (ICI, Coca Cola), INTERCOM (Bell Telephone, Midland Bank) or ADABAS (1000+ installations).

There are some tools that claim the misleading tag 'fourth generation' which cannot claim to be anything new despite part of a new generation. The data base query languages are one group. These provide extra facilities to existing systems allowing rapid enquiry on a database. They do not replace the need for another language or tool to produce cheques, input batch data (transaction processing) or other general data processing requirements. The packages of tools are another. These are a group of tools that together provide all the facilities required but by using at least two separate approaches to the general problem. They do not show thought leading to an integrated design to provide a genuine fourth generation solution.

Thus, of the 65 requests made for information 21 replied. Of these 5 were query languages (not fourth generation contrary to frequent claims), 2 were sets of tools, 1 was an expert system and 1 was a hardware/software performance and tuning package. The remaining three fifths provided 9 4GL's, 2 3GL generators and 1 tool that is almost entirely compilation free.

new language - VISTA is an example of this and has already been discussed above. This is a true fourth generation language. Those at the top of the heap come with database, data dictionary and auto-documentation e.g. ADS/ONLINE.

For example, a POWERHOUSE data dictionary field:

figure 4.11 - Example POWERHOUSE record definition

```
MODE:F  ACTION:ITEM                RECORD Screen

Ø1 Record  ORDERS                                  Auto Defn: M
Ø2 File    ORDERS      Organization  INDEXED       Type RMS
   Open    ORDERS

MODE:F  ACTION:_  Element Attributes   |   Item Attributes

   Element          Usage      Type Siz Dc|Datatype  Size Occ Key Seq.No
Ø1 ORDER-NUMBER     NUMERIC-ID  N    6    |INTEGER-S   4   1   U   1.ØØØ
Ø2 DEPT             ID          C    4    |CHARACTER   4   1   R   2.ØØØ
Ø3 SUPP-CODE        ID          C    4    |CHARACTER   4   1       3.ØØØ
Ø4 GOODS            NAME        C   2Ø    |CHARACTER  2Ø   1       4.ØØØ
Ø5 COST             MONEY       N    9  2 |INTEGER-S   4   1       5.ØØØ
Ø6 DATE-ORDERED                 D    6    |PHDATE      2   1       6.ØØØ
Ø7 DELIVERY-DATE                D    6    |PHDATE      6   1       7.ØØØ
Ø8                                        |
Ø9                                        |
1Ø                                        |
```

Interestingly, the data dictionary identifies a substantial subset of the parameters used to produce a file maintenance program.

producing 3GL code - DELTA [31] is a 3GL code producer. As such it is the nearest to FMG/TPG (the generators produced as part of this study) identified.

DELTA's credo is built into its end product 'DELTA generates portable source COBOL and PL/1 programs' [31/46]. This is backed up by two reasons, (i) outward/forward compatible ' ... expand existing systems ... no-risk long-term software' [31/46] (ii) inward/backward compatible 'existing knowledge of dp staff ... 8Ø% of existing

software is written in COBOL' (31/46)

This is an argument that no doubt the writers of s-algol and ps-algol will be well aware of, powerful despite the ugliness of the product:

figure 4.12 - DELTA screen definition [31/111]

```
.LAYOUT-KW36

  Address-Number: & _______

  Company name      &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

  Credit Limit      _________

  Hard/Soft-Codes   &&&& &&&&      &&&& &&&&

                    &&&& &&&&      &&&& &&&&

                    &&&& &&&&      &&&& &&&&

                    &&&& &&&&

 Next Address Number : & ______              Next Page : __/__  &

.VARIABLES, PREFIX ITS
 -PROGID          , DEFAULT PROGID,      PROTECTED,  NOPAD,  NORMAL
 -DATE,             DEFAULT RUN-EDATE,   PROTECTED,  NOPAD,  NORMAL
 -ACTPAGE         , PROTECTED, NOPAD, NORMAL, JUSTIFY LEFT
 -FUNCTION
 -COMP-NR1,         FROM, PROTECTED
 -COMP-NR2,         FROM, PROTECTED, PIC 9(5)
 -NAME.             FROM, PROTECTED, NOPAD, REVERSE, NORMAL
 -CREDIT-LIMIT,     FROM, TO, PIC Z(5).99.,  MUST,  JUSTIFY LEFT
 TABLE-EQUPMT, TIMES 5
 -TYPE-HARD,        FROM, TO
 -TYPE-SOFT,        FROM, TO, MUST
 TABLE-EQUPMT-END
 -KEY-COMP-NR1
 -KEY-COMP-NR2   , PIC 9(5)
 -NEXTPAGE
 -TOTPAGES        , PROTECTED, NOPAD, NORMAL, JUSTIFY LEFT
 -LASTPOS, MUST, DEFAULT '*'
.END
```

Nonetheless, the actual basic commands isolated in FMG are also present, as are many other unnecessary ones.

in-core, no compilation - perhaps the ultimate tool is one that involves no coding. Although PRO-IV [26] appears to claim this, it requires unique codings to be produced using a 'BASIC-like code' [27].

Despite this, PRO-IV is as far from a language as currently exists according to the survey. Applications are divided into menu, screen, update and report (similar to VISTA's divisions).

PRO-IV has identified the parameters required to produce a file maintenance program as was done for FMG. For example, to define the file maintenance program [27/28ff] requires: mode (update,inquire,etc.), file name, key length, description (file title), sequence number, data type, variable name, maximum length, fill code (space/zero fill alphanumerics), display code (print mask), special check and help message. For example:

figure 4.13 - Part of a PRO-IV Screen Field Definition [27/66]

```
08/16/85        SCREEN FIELD DEFINITION          PCS/SYS/01

    FUNCTION NAME: TRAN-ENT     TRANSACTION ENTRY SCREEN
MODE:
.....................................................................
                                  FIL              SPECIAL GEN M N D A O
FLD ----NAME---- LN COL LEN       CDE DISPLAY-CODE CHECK   CHK I C O R V
... ............ .. ... ... ... ... ............. .....  ... . . . . .
001 BANK-ACCT     8  35  10                                   Y
   BANK ACCOUNT NUMBER
002 BANK-NAME     8  50  30                                       Y
   NAME OF BANK
003 BANK-BRANCH   9  50  30                                       Y

004 @PSEQ         1   1   4          4.0                          Y Y Y

005 BANK-TYPE     1  10   1                                  1   Y   Y Y
   TRANSACTION TYPE ('C'heck, 'D'eposit, 'A'djustment)
006 BANK-REF      1  15   6     RFB                             Y   Y
   TRANSACTION REFERENCE NUMBER (CHECK, DEPOSIT, ADJUSTMENT NOS.)
007 BANK-DATE     1  22   8          DATE          DATE             Y Y
   TRANSACTION DATE
008 BANK-PAY      1  32  30
   REFERENCE NAME (PAYEE FOR CHECKS, FROM WHOM FOR DEPOSITS)
009 BANK-AMT      1  64  15          9,2-
   TRANSACTION AMOUNT
010 BANK-WHAT     2  32  30
   PURPOSE OF TRANSACTION
011 BANK-RECON    2  65   8          DATE          DATE
   DATE OF STATEMENT
```

Again, the minimal set has not been identified as the tool requires input of row and column number amongst others. Superfluous details which can be calculated more logically by a good tool. But PRO-IV uses approximately the same approach as FMG and TPG. "Commercial programming consists largely of the repititious use of a limited number of routines and ... these routines could all be written efficiently in a parameterized manner". [26]

PRO-IV produces auto documentation as does FMG.

Aiming Forward

This variability shows that none of these products can really claim to be what COBOL is relative to the 2GL's. None is as well designed as s-algol or as universal as COBOL (although all are immensely more complex). None show the facets of being properly thought through to avoid (i) the asking of unnecessary questions such as where a field should be shown on a data entry screen, as FMG avoids asking, or (ii) if a new language is required, should it not be as well designed and structured as the best 3GL's, such as the algols?

But the reason the 4GL tools are at least partially succeeding in the extremely conservative commercial language market place is because they are far more powerful than any 3GL and are doing the job commerce requires of them. Even if with less than complete success. Perhaps, herein lies an opportunity? ps-algol, s-algol and FMG evidence what can be done with careful design.

An algol could not take my personal preferred route as used by FMG and TPG. This relies on the present strength and universality of the COBOL market-place. I prefer this as the most fitting short term solution, having used a true fourth generation language, experienced its limitations and having tapped its power into COBOL.

Neither does algol compare with the radically different philosophy of the true fourth generation languages.

But, there is one route left - that taken currently solely in the survey by PRO-IV. This is a tool written in a 3gl that requires any extra facilities to be coded in a 3gl ('BASIC-like'). This tool requires little compilation as the central facility is parameter driven performing file maintenance or printouts as required by the

user and as already defined on a question and answer session.

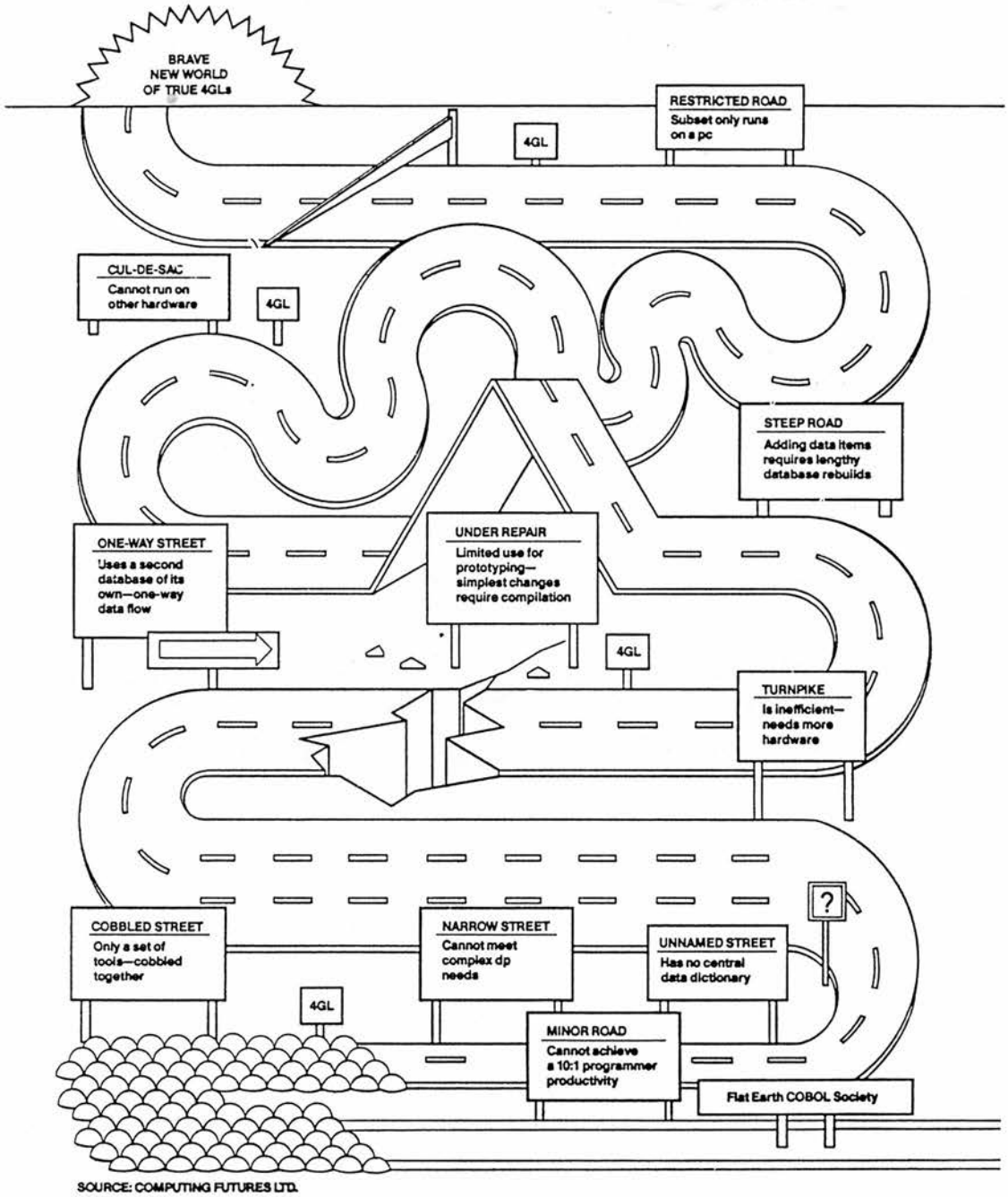
S-algol has already been noted for its clean object code and its ease of coding. With careful thought such a tool could be coded efficiently in s-algol. In addition the extra routines that may be required by the programmer could be coded in the standard format of external procedures and linked into the full suite, as the externals were similarly used in the Byre Theatre bookings system.

Such a task would require a considerable grounding and knowledge in both algol and commercial systems and programming, but would potentially place an algol at the front of the market as a non-compilation-bound facility that is system-consistent and easily expanded and maintained without the recourse to recompilation. It could provide the most advanced product currently envisaged.

Could algol lead the pack in commercial use as clearly as it does in design?

As a final thought, consider figure 4.14

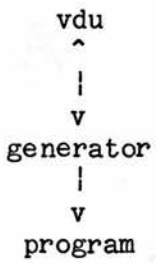
figure 4.14 - 'voyage of discovery' [38,42]



2. ps-algol file maintenance generator

This is a very simple suite that served to prove the concept of s-algol and ps-algol in the commercial fourth generation.

figure 4.15 - ps-algol fmg suite overview



Thus the generator asks each field to be defined. Then the generator produces the fm source program.

This is the very simplest form of generator performing no input checks for correct type and allowing no backtracking to correct errors.

The parameters used are:

(i) file - name, password

(ii) field - name, key, type, occurs, occurs depending on field name, display name, valid values, range, size.

FM Generator

Note that there are 15 - 16 pages relevant to the generator. The COBOL equivalent (see section 3 below) is 35 pages, 125% longer. Apart from the ease of s-algol over COBOL as a coding tool, s-algol thus requires less code.

```
!this is the ps-algol file maintenance program producer
!=====================================================
!
!
!
!
!
structure st.fd(string file.name,key,password;pntr file.fields)
!
structure st field(string code.name, type, depending, short.name;
  int line, page, occurrs; pntr values, size, range, first.field,
  next.field)
!
structure st.s value(string s.value; pntr first.s.value, next.s.value)
structure st.i.value(int i.value; pntr first.i.value, next.i.value)
structure st.r.value(real r.value; pntr first.r.value, next.r.value)
!
structure st.i.range(int i.low, i.high)
structure st.r.range(real r.low, r.high)
!
structure st.i.size(int i.size)
structure st.r.size(real r.size.1, r.size.2)
!
procedure get.file.details(->pntr)
begin
  write "file name"
  let fn:=read.a.line
  write "file password"
  let ps:=read.a.line
  let field:=st.field("","","","",0,0,0,nil,nil,nil,nil,nil)
  let n.page:=1
  let n.line:=5
  let f.pg:=0
  let f.ln:=0
  let f.ke:=""
  let done:=false
  while ~done do
  begin
    write "field name (c/r to end fields)"
    let f.fn:=read.a.line
    if f.fn="" then done:=true else
    begin
      if f.ke="" do
      begin
        write "key? (y=yes)"
```

```
 if read.a.line="y" then f.ke:=f.fn
end
write "type (i/r/s/t/d) (upper case only)"
let f.ty:=read.a.line
write "occurs (1 if norm else >1)"
let f.oc:=readi
let f.de:=""
if f.oc>1 do
begin
  write "depending on field"
  f.de:=read.a.line
end
if f.ke=f.fn then
begin
  f.pg:=1
  f.ln:=3
end
else
begin
  if n.line+f.oc>23 then
  begin
    n.page:=n.page+1
    f.pg:=n.page
    f.ln:=5
    n.line:=4:f.oc
  end
  else
  begin
    f.ln:=n.line
    n.line:=n.line+f.oc
    f.pg:=n.page
  end
  if n.line=23 do
  begin
    n.line:=5
    n.page:=n.page+1
  end
end
write "short name (for displaying on the vdu)"
let f.sh:=read.a.line
let value:=(case f.ty of
"i":st.i.value(0,nil,nil)
"r","t":st.r.value(0.0,nil,nil)
"s":st.s.value("",nil,nil))
default:nil
let a.value:=false
let f.done:=false
if f.ty~="d" do while ~f.done do
begin
  write "enter a valid value? (n=no more)"
  if (read.a.line)="n" then f.done:=true else
  begin
    a.value:=true
    value:= (case f.ty of
    "i":st.i.value(readi,(if value(first.i.value)=nil then
      value else value(first.i.value)),nil)
    "r","t":st.r.value(readr,(if value(first.r.value)=nil then
      value else value(first.r.value)),nil)
    default:st.s.value(read.a.line,(if value(first.s.value)=nil
```

```
          then value else value(first.s.value),nil))
        value:=(case f.ty of
        "i","d":value(next.i.value)
        "r","t":value(next.r.value)
        default:value(next.s.value))
      end
    end
    value:=(if a.value then (case f.ty of
      "i":value(first.i.value)
      "r","t":value(first.r.value)
      default value(first.s.value)) else nil)
    if f.ty~="s" do if f.ty~="d" do write "range (lower and upper)"
    let range:=(case f.ty of
    "i","d":st.i.range(readi,readi)
    "r","t":st.r.range(readr,readr)
    default:nil)
    if f.ty="s" or f.ty="i" do write "size"
    if f.ty="r" do write "size (before and after the point)"
    let size:=(case f.ty of
    "i","s":st.i.size(readi)
    "r":st.r.size(readr,readr)
    "t":st.r.size(2,2)
    default:st.i.size(6))
    field:=st.field(f.fn,f.ty,f.de,f.sh,f.pg,f.ln,f.oc,value,size,
      range,(if field(first.field)=nil then field else
      field(first.field),nil)
    field:=field(next.field)
  end
end
field:=field(first.field)
st.fd(fn,key,password,field)
end
!
procedure new.page.code(pntr fds;file program)
begin
  let n.page:=0
  output(program),"let new.page=proc(int page)
begin
  for i=5 to 22 do
  begin
    cursor.to(i,1)
    cursor("erase")
  end
  case page of
"
  let field:=fds(file.fields)
  while field~=nil do
  begin
    if field(page)~=n.page do
    begin
      n.page:=field(page)
      output(program),
"  "++iformat(n.page)++
":
  begin
"
    end
    output(program),
"  display."++
```

```
      field(code.name)++"("++field(file.name)++"("++
      field(code.name)++
"))
"
    field:=field(next.field)
    if field=nil or n.page~=field(page) do output(program),
"  end
"
  end
  output(program),
"  default:fail('"invalid page number at new.page='"++iformat(page))
end
!
"
end
!
procedure check.code(pntr fds;file program)
begin
  output(program),"!
!input checking
!
"
  let next:=fds(file.fields)
  while next~=nil do
  begin
    output(program),"let check."++next(code.name)++"=proc("++
      (case next(type) of
        "i","d":"int "
        "r","t":"real "
        default:"string ")++next(code.name)++" -> bool)
begin
"
    let xvalues:=next(values)
    let xrange:=next(range)
    let xsize:=next(size)
    if xvalues=nil and xrange=nil then output(program),
"  true
"
    else
    begin
      if xvalues~=nil do
      begin
        output(program),
"  case "++
          next(code.name)++
" of
    "
        while xvalues~=nil do
        begin
          output(program),(if next(type)="s" then
"'"""
            else
""")++
            (case next(type) of
            "i","d":xvalues(i.value)
            "r","t":xvalues(r.value)
            default:xvalues(s.value))++(if next(type)="s" then
"'"""
            else
```

```
""")++
          xvalues:=xvalues(next.value)
          if xvalues~=nil do output(program),
","
        end
        output(program),
":true
    default:"
      end
      output(program),(if xrange=nil then
"false
"
        else
"  if "++
        next(code.name)++
" >= "++
        (case next(type) of
        "i","d":iformat(xrange(i.low))
        default:fformat(xrange(r.low),xsize(r.size.1),
        xsize(r.size.2))++
" and "++
        next(code.name)++
" <= "++
        (case next(type) of
        "i","d":iformat(xrange(i.high))
        default:fformat(xrange(r.high),xsize(r.size.1),
        xsize(r.size.2))++
"
    then true else false
"
    end
    output(program),
"end
!
"
    next:=next(next.field)
  end
end
!
procedure input.code(pntr fds;file program)
begin
  output(program),
"!
!input and verify
!
"
  let next:=fds(file.fields)
  while next~=nil do
  begin
    let xsize:=next(size)
    let size:=(case next(type) of
    "r","t":xsize(r.size.1)+xsize(r.size.2)+2
    default:xsize(i.size))
    let xrange:=next(range)
    output(program),
"let input."++
      next(code.name)++
"=proc( -> "++
      (if next(occurs) > 1 then
```

```
"*"
        else
"")++
      (case next(type) of
        "i","d":"int"
        "r","t":"real"
        default:"string")++
")
begin
  let ok:=false
  let "++
      next(code.name)++
":="++
      (if next(occurs)>1 then
"vector 1::"++
        iformat(next(occurs))++
" of "
        else
"")++
      (case next(type) of
        "i","d":"0"
        "r","t":"0.0"
        default:"'""'")++
"
"++
      (if next(occurs)>1 then
"  for i=1 to "++
        iformat(next(occurs))++
" do
  begin
"
        else
"")++
"  while ~ok do
  begin
    prints("++
      (if next(occurs)>1 then iformat(next(line)-1)++"+i" else
 iformat(next(line))++
",20,'""
    for i=1 to size do output(program),
"_"
    output(program),
"'")
    "++
      next(code.name)++
      (if next(occurs)>1 then
"(i)"
        else
"")++
":=input"++
      (case next(type) of
        "i","d":"i"
        "r","t":"r"
        default:"s")++
"("++
      (if next(occurs)>1 then iformat(next(line)-1)++"+i" else
 iformat(next(line))++
",20,'""
      (case next(type) of
```

```
        "i","d":iformat(xrange(i.low))++
","++
          iformat(xrange(i.high))
        "r","t":fformat(xrange(r.low),xsize(r.size.1),xsize(r.size.2))++
","++
          fformat(xrange(r.high),xsize(r.size.1),xsize(r.size.2))
        default:iformat(size))++
")
    ok:=check."++
      next(code.name)++
"("++
      next(code.name)++
      (if next(occurs)>1 then
"(i))
  end
  ok:=false
"
        else
"")++
"  end
  "++
      next(code.name)++
"
end
!
"
    next:=next(next.field)
  end
end
!
procedure display.code(pntr fds;file program)
begin
  output(program),
"!
!display
!"
  let next:=fds(next.field)
  while next~=nil do
  begin
    output(program),
"let display."++
      next(code.name)++
"=proc("++
      (if next(occurs)>1 then
"*"
        else
"")++
      (case next(type) of
        "i","d":"int "
        "r","t","real "
        default:"string ")++
      next(code.name)++
")
begin
  "++
      (if next(occurs)>1 then
"for i = 1 to "++
        iformat(next(occurs))++
" do
```

```
  begin
    "
        else
""")++
"  print"++
      (case next(type) of
      "i","d":"i"
      "r","t":"r"
      default:"s")++
"("++
","++
      (if next(occurs)>1 then iformat(next(line)-1)++"+i" else
      iformat(next(line)))++
",20,"++
      next(code.name)++
      (if next(occurs)>1 then
"(i)"
        else
""")++
")
"++
      (if next(occurs)>1 then
"  end
  "
        else
""")++
"end
!
"
    next:=next(next.field)
  end
end
!
procedure audit.code(pntr fds;file program)
begin
  output(program),
"!
!audit the record
!
let audit.print=proc(pntr "++
    fds(file.name)++
";file audit)
begin
  "
  let next:=fds(next.field)
  while next~=nil do
  begin
    let offset:=iformat(20-length(next(short.name)))
  output(program),
    (if next(occurs)>1 then
"  for i=1 to "++
      iformat(next(occurs))++
" do
  begin
    output(audit),(if i=1 then
'""++
      next(short.name)++
"'" else '"'"),
    "++
```

```
      next(code.name)++
"(i):(if i=1 then "++
      offset++
" else 20),'"
'"
  end
"
  else
"  output(audit),'""++
      next(short.name)++
"'",'"++
      fds(file.name)++
"("++
      fds(file.name)++
".'"++
      next(code.name)++
"):"++
      offset++
",'"
'"")
    next:=next(next.field)
  end
  output(program),
"end
!
"
end
!
procedure screen.background.code(pntr fds;file program)
begin
  output(program),
"!
!display screen background
!
let screen.background=proc()
begin
cursor('"erase'")
  prints(1,1,'"University of St. Andrews'")
  prints(1,70,'""++
    fds(program.name)++
"'")
end
!
"
end
!
procedure get.function.code(pntr fds;file program)
begin
  output(program),
"!
!get.function
!
let get.function=proc(->string)
begin
  let ok:=false
  let function:='"'"
  prints(23,1),'"amend, show, insert or delete (a/s/i/d)'"
  while ~ok do
  begin
```

```
    function:=inputs(23,42,1)
    ok:=(case function of
      '"'",'"a'",'"s'",'"i'",'"d'":true
      default:false
  end
  cursor.to(2,70)
  cursor('"erase'")
  prints((case function of
    '"a'":'"amend'"
    '"s'":'"show'"
    '"i'":'"insert'"
    '"d'":'"delete'"
    default:'"'"),2,70)
  cursor.to(23,1)
  cursor('"erase'")
  function
end
!"
end
!
procedure get.key.code(pntr fds;file program)
begin
  let n.key:=fds(key)
  let next:=fds(next.field)
  let ok:=0
  while next~=nil do
    if next(code.name)=n.key then ok:=1
    else next:=next(next.field)
  if ok=0 do fail("get.key.code - key not found")
  output(program),
"!
!get key and record details
!
let get.key=proc(pntr "++
    fds(file.name)++
","++
    fds(file.name)++
".list -> int)
begin
  let ok:=0
  let record:=nil
  let key:="++
    (case next(type) of
      "i","d":"0"
      "r","t":"0.0"
      default:"'"'"")++
  while ok=0 do
  begin
    key:=input."++
    next(code.name)++
"()
    if key="++
    (case next(type) of
      "i","d":"0"
      "r","r":"0.0"
      default:"'"'"")++
" then ok:=2 else
    begin
      record:="++
```

```
(case next(type) of
  "i","d":"i"
  "r":"r"
  default:"s")++".lookup(key,"++
    fds(file.name)++
".list)
      ok:=(if fd.list=nil then (if function='"i'" then 1 else 0)
        else (if function~='"i'" then 1 else 0))
      if ok=0 do call error('"record does'"++(if function='"i'"
        then '"'" else '" not '")++'"exist'")
    end
  end
  ok
end
!
end
!
procedure show.function.code(pntr fds;file program)
begin
  let n.page:=0
  output(program),
"!
!show.function
!
procedure show.function(pntr "++
    fds(file.name)++
")
begin
  "
  let next:=fds(next.field)
  while next~=nil do
  begin
    if n.page~=next(page) do
    begin
      if n.page>0 do output(program),
"  message('"press return for next page'")
"
      n.page=next(n.page)
      output(program),
"  new.page("++
        iformat(n.page)++
")
"
    end
    output(program),
"  display."++
      next(code.name)++
"("++
      fds(file.name)++
"("++
      fds(file.name)++
"."++
      next(code.name)++
"))
"
    next:=next(next.field)
  end
  output(program),
"end
```

```
!
"
end
!
procedure amend.function.code(pntr fds;file program)
begin
  let n.page:=0
  output(program),
"!
!amend function
!
procedure amend.function (pntr "++
    fds(file.name)++
")
begin
  let x:=1
  while x~=0 do
  begin
    x:=inputi.23(1,200)
    case x of "
  let count:=1
  let next:=fds(next.field)
  while next~=nil do
  begin
    if n.page~=next(page) do
    begin
      output(program),
      iformat(count)++
":"++
"  begin
    new.page("++
        iformat(next(page))++
")
"
    end
    output(program),
"     "++
      fds(file.name)++
"("++
      fds(file.name)++
"."++
      next(field.name)++
"):=input."++
      next(field.name)++
"
"
    if n.page~=next(page) do
    begin
      n.page:=next(page)
      if n.page~=0 do output(program),
"  end
"
    end
    count:=count+1
    next:=next(next.field)
  end
  output(program),
"    default:{}
  end
```

```
end
!
"
end
!
procedure input.function.code(pntr fds;file program)
begin
  let n.page=0
  output(program),
"!
!input function
!
procedure input.function (pntr "++
    fds(file.name)++
")
begin
"
  let next:=fds(next.field)
  while next~=nil do
  begin
    if n.page~=next(page) do
    begin
      n.page=next(n.page)
      output(program),
"  new.page("++
        iformat(n.page)++
")
"
    end
    output(program),
"  "++
      fds(file.name)++
"("++
      fds(file.name)++
"."++
      next(code.name)++
"):=input."++
      next(code.name)++
"
"
  end
  output(program),
"  amend.function("++
    fds(file.name)++
")
end
!
"
end
!
procedure main.code.code(pntr fds;file program)
begin
  let n.key:=fds(key)
  let next:=fds(next.field)
  let ok:=0
  while next~=nil do
    if next(code.name)=n.key then ok:=1
    else next:=next(next.field)
  if ok=0 do fail("get.key.code - key not found")
```

```
  output(program),
"!
!main program
!============
screen.background()
let function:='"a'"
let "++
    fds(file.name)++
":=open.database('""++
    fds(file.name)++
".db,'""++
    password++
"'",'"write'")
if "++
    fds(file.name)++
" is error.record do fail('"failed to open "++
    fds(file.name)++
" database'")
let "++
    fds(file.name)++
".list:=s.lookup('""++
    fds(file.name)++
"'",'"++
    fds(file.name)++
")
if "++
    fds(file.name)++
".list=nil do
begin
  "++
    fds(file.name)++
".list=table()
  s.enter('""++
    fds(file.name)++
"'",'"++
    fds(file.name)++
",'"++
    fds(file.name)++
".list)
end
let audit:=create('""++
    fds(file.name)++
".log'",493)
if audit=nullfile do fail('"failed to create log'")
audit:=open('""++
    fds(file.name)++
".log'",1)
if audit=nullfile do fail('"failed to open log'")
!
!audit headings
!
output(audit),'"University of St Andrews'",'""++
    fds(file.name)++
"':70,'""++


'"
!
while function~='"'" do
```

```
begin
  function:=get.function()
  if function~='""' do
  begin
    let key:="++
    (case next(type) of
      "i","d":"1"
      "r","t":"1.0"
      default:"'""'"")++
"
    while key~="++
    (case next(type) of
      "i","d":"maxint"
      "r","t":"maxreal"
      default:"'""'"")++
"
    do
    begin
      let flag:=get.key("++
      fds(file.name)++
","++
      fds(file.name)++
".list,function)
      if flag<2 do case function of
      '"i'": !input a new record
      begin
        output(audit),'"insert a new record

"
        input.function("++
      fds(file.name)++
")
        if inputb.23('"all details correct'",'"y'") then
        begin
          "++
      (case next(type) of
        "i","d":"i"
        "r":"r"
        default:"s")++
".enter(key,"++
      fds(file.name)++
","++
      fds(file.name)++
".list)
          audit.print("++
      fds(file.name)++
",audit)
        end
        else output(audit,'"abandoned


'"
      end
      '"s'": ! show an existing record
      begin
        output(audit),'"show a record

'"
        show.function("++
```

```
      fds(file.name)++
")
        audit.print("++
      fds(file.name)++
",audit)
      end
      '"a'": ! amend an existing record
      begin
        output(audit),'"amend an existing record

'"
        audit.print("++
      fds(file.name)++
",audit)
        show.function("++
      fds(file.name)++
")
        amend.function("++
      fds(file.name)++
")
        if inputb.23('"all details correct'",'"y'") then
        begin
          "++
      (case next(type) of
        "i","d":"i"
        "r":"r"
        default:"s")++
".enter(key,"
      fds(file.name)++
","++
      fds(file.name)++
".list)
          audit.print("++
      fds(file.name)++
",audit)
        end
        else output(audit),'"abandoned


'"
      end
      default: ! delete an existing record
      begin
        output(audit),'"delete a record

'"
        show.function("++
      fds(file.name)++
")
        audit.print("++
      fds(file.name)++
",audit)
        if inputb.23('"delete this record'",'"y'") then
        begin
          "++
      fds(file.name)++
".list:=nil
          "++
      (case next(type) of
```

```
        "i","d":"i"
        "r":"r"
        default:"s")++
".enter(key,"
      fds(file.name)++
","++
      fds(file.name)++
".list)
        end
        else  output(audit),'"abandoned


'"
      end
    end
  end
end
output(audit),'"end of audit report''p'"
close(audit)
let committed = commit()
if committed is error.record do fail('"failed to cimmit "++
   fds(file.name)++
" database'")
?
"
end
!
procedure start.code(pntr fds;file program)
begin
  output(program),
"!"++
    fds(program.name)++
" file maintenance program
!
!
!================================================================
!
!
"
  output(program),
"structure st."++
      fds(file.name)++
"("
  let fields:=fds(file.fields)
  let count:=2
  let n.type:=""
  while fields~=nil do
  begin
    output(program),
      (if field(occurs)>1 then
"*"
        else
"")++
      (if n.type~=fields(type) then (case fields(type) of
          "i","d":"int "
          "r","t":"real "
        default:"string ") else "")++
      fields(code.name)
    if count=4 then
```

```
    begin
      output(program),
"
"
      count:=1
    end
    else count:=count+1
    fields:=fields(next.field)
    if fields~=nil then
    begin
      output(program),(if fields(type)~=n.type then
";"
        else
",")
      n.type:=fields(type)
    end
    else output(program),
")
!
"
  end
  output(program),
"!
!================================================================
!
"
end
!
!main program code for the file maintenance producer
!=================================================
!
let fds:=get.file.details
if fds~=nil do
begin
  let program:=create(fds(file.name)++".s","s","a","v",133)
  if program=nullfile do fail("couldn''t create program file")
  program:=open(fds(file.name)++".s","a",1)
  if program=nullfile do fail("couldn''t open program file")
  new.page.code(fds,program)
  start.code(fds,program)
  check.code(fds,program)
  input.code(fds,program)
  display.code(fds,program)
  audit.code(fds,program)
  screen.background.code(fds,program)
  get.function.code(fds,program)
  get.key.code(fds,program)
  show.function.code(fds,program)
  amend.function.code(fds,program)
  input.function.code(fds,program)
  main.code(fds,program)
  close(program)
end
```

ps-algol Output Source

Although the s-algol fm generator differs slightly from the COBOL fm generator (see section 3 below) the complete fm program below is less than 6 pages. An equivalent COBOL program is 15 pages, 150% longer.

```
!nomacc file maintenance program
!
!
!================================================================
!
!
structure st.nomacc( string nomacc.status; int nomacc.account;
*string nomacc.address)
!
!
!================================================================
!
!
!input checking
!
let check.status=proc(string status -> bool)
begin
  case status of
    " ","X":true
    default:false
end
!
let check.account=proc(int account -> bool)
begin
  if account >= 000010 and account <= 999999
    then true else false
end
!
let check.address=proc(string address -> bool)
begin
  true
end
!
!
!input and verify
!
let input.status=proc( -> string)
begin
  let ok:=false
  let status:=""
  while ~ok do
  begin
    status:=inputs(5,20,1)
    ok:=check.status(status)
  end
```

```
  status
end
!
let input.account=proc( -> int)
begin
  let ok:=false
  let account:=0
  while ~ok do
  begin
    account:=inputi(3,20,6,0)
    ok:=check.account(account)
  end
  account
end
!
let input.address=proc( -> *string)
begin
  let ok:=false
  let address:=vector 1::5 of ""
  for i=1 to 5 do
  begin
  while ~ok do
  begin
    address(i):=inputs(5+i,20,30)
    ok:=check.address(address(i))
  end
  ok:=false
  end
  address
end
!
!
!display
!
let display.status=proc(string status)
begin
    prints(5,20,status)
end
!
let display.account=proc(int account)
begin
    printi(3,20,account)
end
!
let display.address=proc(*string address)
begin
  for i = 1 to 5 do
  begin
      prints(5+i,20,address(i))
  end
end
!
!
let new.page=proc(int page;pntr nomacc)
begin
  for i=5 to 22 do
  begin
    cursor.to(i,1)
    cursor("erase")
```

```
  end
  case page of
  1:
  begin
  printi(5,1,1)
  prints(5,5,"record status")
  display.status(nomacc(nomacc.status))
  printi(6,1,2)
  prints(6,5,"name and address")
  display.address(nomacc(nomacc.address))
  end
  default:fail("invalid page number at new.page="++iformat(page))
end
!
!audit the record
!
let audit.print=proc(pntr nomacc;file audit)
begin
    output(audit),"record status",nomacc(nomacc(nomacc.status)):7,"
"  output(audit),"ledger account",nomacc(nomacc(nomacc.account)):6,"
"  for i=1 to 5 do
  begin
    output(audit),(if i=1 then "name and address" else ""),
    nomacc(nomacc.address)(i):(if i=1 then 4 else 20),"
"
  end
end
!
!display screen background
!
let screen.background=proc()
begin
  cursor("erase")
  prints(1,1,"university of St. Andrews")
  prints(1,70,"nomacc")
end
!
!
!get.function
!
let get.function=proc(->string)
begin
  let ok:=false
  let function:=""
  prints(23,1,"amend, show, insert or delete (a/s/i/d)")
  while ~ok do
  begin
    function:=inputs(23,42,1)
    ok:=(case function of
      "","a","s","i","d":true
      default:false
  end
  cursor.to(2,70)
  cursor("erase")
  prints(2,70,(case function of
    "a":"amend"
    "s":"show"
    "i":"insert"
    "d":"delete"
```

```
    default:""))
  cursor.to(23,1)
  cursor("erase")
  function
end
!
!
!get key and record details
!
let get.key=proc(pntr nomacc,nomacc.list -> int)
begin
  let ok:=0
  let record:=nil
  let key:=0
  while ok=0 do
  begin
    key:=input.account()
    if account=0 then ok:=2 else
    begin
      record:=i.lookup(key,nomacc.list)
      ok:=(if fd.list=nil then (if function="i" then 1 else 0)
        else (if function~="i" then 1 else 0))
      if ok=0 do call error("record does"++(if function="i" then ""
        else " not ")++"exist")
    end
  end
  if ok=1 do nomacc(nomacc.account):=key
  ok
end
!
!
!show.function
!
let show.function=proc(pntr nomacc)
begin
  new.page(1,nomacc)
  display.status(nomacc(nomacc.status))
  display.account(nomacc(nomacc.account))
  display.address(nomacc(nomacc.address))
end
!
!
!amend function
!
let amend.function=proc(pntr nomacc)
begin
  let x:=1
  while x~=0 do
  begin
    x:=inputi.23(1,200)
    case x of
  1:begin
    new.page(1,nomacc)
    nomacc(nomacc.status):=input.status()
  end
  2:  nomacc(nomacc.account):=input.account()
  3:  nomacc(nomacc.address):=input.address()
   default:{}
  end
```

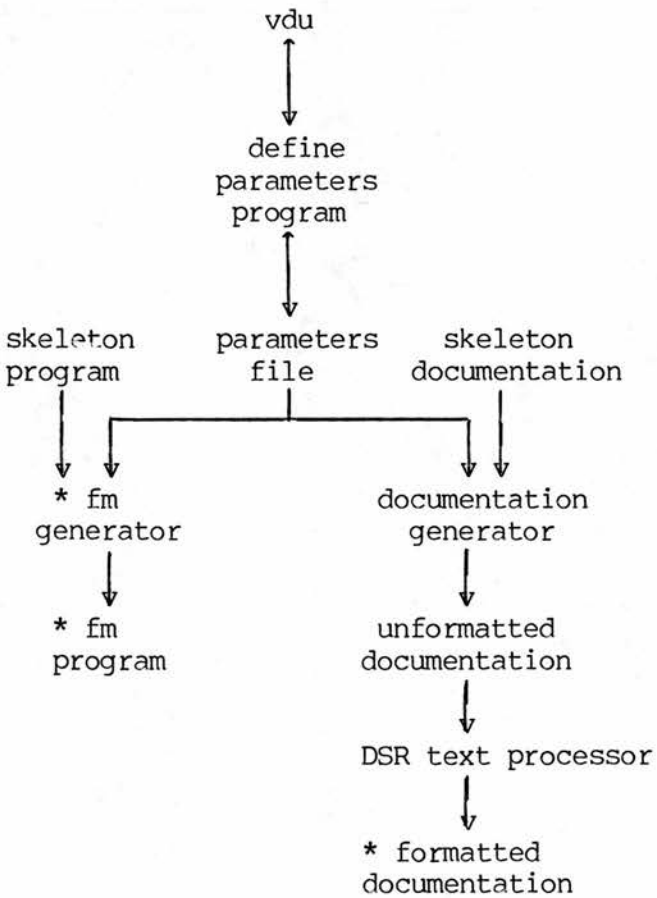
```
end
!
!input function
!
let input.function=proc(pntr nomacc)
begin
  new.page(1,nomacc)
  nomacc(nomacc.status):=input.status()
  nomacc(nomacc.account):=input.account()
  nomacc(nomacc.address):=input.address()
  amend.function(nomacc)
end
!
!main program
!============
screen.background()
let function:="a"
let nomacc:=open.database("nomacc.db,"resurgam","write")
if db is error.record do fail("failed to open nomacc database")
let nomacc.list:=s.lookup("nomacc",nomacc)
if nomacc.list=nil do
begin
  nomacc.list:=table()
  s.enter("nomacc",nomacc,nomacc.list)
end
let audit:=create("nomacc.log",493)
if audit=nullfile do fail("failed to create log")
audit:=open("nomacc.log,1)
if audit=nullfile do fail("failed to open log")
!
!audit headings
!
output(audit),"university of St. Andrews","nomacc":70,"


"
!
while function~="" do
begin
  function:=get.function()
  if function~="" do
  begin
    let key:=1
    while key~=maxint
    do
    begin
      let flag:=get.key(nomacc,nomacc.list,function)
      if flag<2 do case function of
      "i": !input a new record
      begin
        output(audit),"insert a new record

"
        input.function(nomacc)
        if inputb.23("all details correct","y") then
        begin
          i.enter(key,nomacc,nomacc.list)
          audit.print(nomacc,audit)
        end
```

```
        else output(audit,"abandoned


"
      end
      "s": ! show an existing record
      begin
        output(audit),"show a record

"
        show.function(nomacc)
        audit.print(nomacc,audit)
      end
      "a": ! amend an existing record
      begin
        output(audit),"amend an existing record

"
        audit.print(nomacc,audit)
        show.function(nomacc)
        amend.function(nomacc)
        if inputb.23("all details correct","y") then
        begin
          i.enter(key,nomacc,nomacc.list)
          audit.print(nomacc,audit)
        end
        else output(audit),"abandoned


"
      end
      default: ! delete an existing record
      begin
        output(audit),"delete a record

"
        show.function(nomacc)
        audit.print(nomacc,audit)
        if inputb.23("delete this record","y") then
        begin
          nomacc.list:=nil
          i.enter(key,nomacc,nomacc.list)
        end
        else  output(audit),"abandoned


"
      end
    end
  end
end
output(audit),"end of audit report'p"
close(audit)
let committed=commit()
if committed is error.record do fail(" failed to commit nomacc database")
?
```

3. The COBOL File Maintenance Generator

This represents only the program producing section of the generator and an example of the program output from the generator. Full details of the VAX DCL command procedure to run the generator, f.m. program definition program, f.m. program producer, documentation producer, documentation and associated files are held in an unbound appendix and tape at the department.

figure 4.16 - COBOL fmg suite overview



* indicates these are included as examples below.

The generator:

(i) asks for the program name. It checks whether there is already a parameter file set up for this program from a previous run and notifies the operator whether it already exists.

(ii) the parameter program asks for the details of the program to be produced. As this is a fm program the operator can alter the fm parameters.

(iii) the generator then produces the fm program defined by the parameters.

(iv) the command procedure then optionally compiles the output program (a useful test on first producing a program from a set of parameters as reserved names, duplicate field names or some other error may be produced from the parameters. This is a range of tests too considerable to put into the generator at this stage).

(v) the the generator produces the system and operating documentation and optionally formats it using DSR (RUNOFF) the DEC VAX VMS text processing package (the documentation is produced in a format to be input to DSR).

At each stage the operator may abandon the run or omit an option.

The parameters are:

(i) file - name, title

(ii) field - name, type, display name, key, duplicates (if field is the primary key), amendable, occurrences, valid values, size, range.

Produce FM program

This is an example that shows how the generator produced the COBOL code. Note that it is far more complex than the s-algol equivalent (see section 2 above). This is partly due to the more rigid structure of the COBOL language making variable positioning of output fields difficult.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. FMTWO.
*
* This program is a tool written that
*   given a file defining a file's field
*   definitions it produces a
*   complete working ANSI COBOL
*   program to run on a DEC VAX under VMS on
*   VT1ØØ terminal. The object program will allow
*   full screen file maintenance with audit
*   of actions produced automatically.
*
* Screen layouts, print layouts and user operating
*   instructions are produced by another
*   program.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT PROG-FILE ASSIGN TO PROG.
     SELECT SKELETON ASSIGN TO SKEL.
     SELECT FM ASSIGN TO FMF
         ORGANIZATION IS INDEXED
         ACCESS IS DYNAMIC
         RECORD KEY IS FM-KEY
         ALTERNATE RECORD KEY IS FM-FIELD.
 DATA DIVISION.
 FILE SECTION.
 FD FM.
 Ø1 FM-RECORD.
     Ø3 FM-KEY                   PIC 9(3).
     Ø3 FM-FILE                  PIC X(9).
     Ø3 FM-TITLE                 PIC X(5Ø).
     Ø3 FM-FIELD                 PIC X(9).
     Ø3 FM-TYPE                  PIC X.
     Ø3 FM-DISPLAY-N             PIC X(2Ø).
     Ø3 FM-KEY-FLAG              PIC 9.
     Ø3 FM-KEY-DUPS              PIC X.
     Ø3 FM-AMEND                 PIC X.
     Ø3 FM-OCCURS                PIC 99.
     Ø3 FM-NUM-VALS              PIC 99.
```

```
    03 FM-TYPE-VARIABLE.
        05 FM-VALUES-S          PIC X(20) OCCURS 15.
        05 FM-SIZE-S            PIC 9(2).
        05 FILLER               PIC X(6).
    03 FILLER REDEFINES FM-TYPE-VARIABLE.
        05 FM-VALUES-I          PIC S9(18) OCCURS 15.
        05 FM-SIZE-I            PIC 9(2).
        05 FM-RANGE-I           PIC S9(18) OCCURS 2.
    03 FILLER REDEFINES FM-TYPE-VARIABLE.
        05 FM-VALUES-R          PIC S9(13)V9(5) OCCURS 14.
        05 FM-SIZE-RB           PIC 9(2).
        05 FM-SIZE-RA           PIC 9(2).
        05 FM-RANGE-R           PIC S9(13)V9(5) OCCURS 2.
        05 FILLER               PIC X(16).
    03 FILLER REDEFINES FM-TYPE-VARIABLE.
        05 FM-VALUES-D          PIC 9(6) OCCURS 15.
        05 FM-SIZE-D            PIC 9(2).
        05 FM-RANGE-D           PIC 9(6) OCCURS 2.
        05 FILLER               PIC X(204).
    03 FM-SCREEN-NU             PIC 9(3).
    03 FM-LINE                  PIC 9(2).
    03 FM-PAGE                  PIC 9(2).
FD  PROG-FILE
        RECORD VARYING FROM 7 TO 79 CHARACTERS
        DEPENDING ON RECORD-LENGTH.
01  OUT-REC.
    03  ORR PIC X OCCURS 79.
FD  SKELETON.
01  SKEL-LONG-REC.
    03  FILLER PIC XXX.
    88  END-OF-DATA VALUE "***".
    03  FILLER PIC X(100).
01  SKEL-REC PIC X(72).
WORKING-STORAGE SECTION.
01  ST-INTEGER            PIC -(17)9.
01  FILLER.
    03  ST-REAL          PIC -(12)9.9(5).
    03  STR REDEFINES ST-REAL PIC X OCCURS 19.
01  PAST-ZERO             PIC 9 COMP.
01  ST-FM-CLOSED          PIC 9 COMP.
01  WORK-1           PIC 99 COMP.
01  WORK-2           PIC 99 COMP.
01  EOF-REPLY PIC X.
    88 END-OF-FILE VALUE "1".
01  RECORD-LENGTH PIC 99 COMP.
01  W-QUOTE PIC X VALUE QUOTE.
01  WORK-A PIC 9999.
01  WORK-B PIC 9999.
01  WORK-C PIC 9999.
01  WORK-D PIC 9999.
01  WORK-E PIC 9999.
01  WORK-F PIC 9999.
01  WORK-G PIC 9999.
01  WORK-K PIC 9999.
01  PPAGE PIC 9999 VALUE 1.
01  KEY-NAME PIC X(9) VALUE SPACES.
01  KEY-TYPE PIC X VALUE SPACE.
01  INDENT PIC 9.
    88 COMMENT VALUE 1.
```

```
           88 AREA-A VALUE 2.
           88 AREA-B VALUE 3.
           88 INDENTED VALUE 4.
           88 SAME-LINE VALUE 5.
           88 NO-INDENT VALUE 6.
       01  CURR PIC 99 VALUE 7.
       01  FIELD.
           03  FI1 PIC X(60) VALUE SPACES.
           03  FI REDEFINES FI1 PIC X OCCURS 60.
       01  FILE-NAME.
           03  FN3 PIC XXX.
           03  FILLER PIC X(6).
       01  ST-VALUES.
           03  VIN PIC X OCCURS 20.
       01  FILE-TITLE PIC X(60).
       PROCEDURE DIVISION.
       START-X SECTION.
       A00.
           PERFORM INITIALX.
           PERFORM MAINFLOW.
           PERFORM CLOSEDOWN.
       A99.
           STOP RUN.
      *
      *
       INITIALX SECTION.
      *
      * open the files
      *
       I00.
           OPEN INPUT FM.
           MOVE 0 TO ST-FM-CLOSED.
           OPEN INPUT SKELETON.
           OPEN OUTPUT PROG-FILE.
       I99.
           EXIT.
      *
      * produce the program
      *
       MAINFLOW SECTION.
       M00.
           PERFORM FIRST-DETAILS.
           PERFORM STATIC-DETAILS.
           PERFORM XINPUT-FIELDS.
           PERFORM XAMEND-FIELDS.
           PERFORM XSHOW-FIELDS.
           PERFORM DISPLAY-STATICS.
           PERFORM XINPUT-FIELD.
           PERFORM XDISPLAY-FIELD.
           PERFORM XNEW-PAGE.
           PERFORM XAUDIT-MASTER.
       M99.
           EXIT.
      *
      * close files
      *
       CLOSEDOWN SECTION.
       C00.
           PERFORM CLOSE-FM.
```

```
     CLOSE PROG-FILE.
     CLOSE SKELETON.
 C99.
     EXIT.
*
 CLOSE-FM SECTION.
 CFM00.
     IF ST-FM-CLOSED = ZERO
         CLOSE FM
         MOVE 1 TO ST-FM-CLOSED.
 CFM99.
     EXIT.
*
* this forces the program to fail with an error being
*  returned to the operating system.
*
 FAIL-END SECTION.
 FA00.
     DISPLAY "PROGRAM RUN ABANDONED" WITH NO ADVANCING.
     MOVE ZERO TO WORK-1.
     DIVIDE 2 BY WORK-1 GIVING WORK-2.
     PERFORM CLOSEDOWN.
     STOP "PROGRAM RUN ABANDONED".
 FA99.
     EXIT.
*
 MAS-HYPHEN SECTION.
 MAS00.
     MOVE FILE-NAME TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE "-" TO FIELD.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
 MAS99.
     EXIT.
*
 STATIC-DETAILS SECTION.
 ID00.
     PERFORM IO-SKEL.
     MOVE "       PROGRAM-ID. FM" TO OUT-REC.
     MOVE 22 TO CURR.
     MOVE FILE-NAME TO FIELD.
     MOVE 6 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE "." TO FIELD.
     PERFORM ADD-FIELD.
     PERFORM IO-SKEL.
     MOVE "           SELECT" TO OUT-REC.
     MOVE 19 TO CURR.
     MOVE FILE-NAME TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "ASSIGN TO" TO FIELD.
     PERFORM ADD-FIELD.
     MOVE FILE-NAME TO FIELD.
     PERFORM ADD-FIELD.
```

```
    SUBTRACT 1 FROM CURR.
    MOVE "F" TO ORR (CURR).
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    MOVE "                RECORD KEY IS" TO OUT-REC.
    MOVE 30 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE KEY-NAME TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM START-DETAILS.
ID01.
    IF FM-KEY-FLAG = 2
        PERFORM WRITEIT
        MOVE "                ALTERNATE RECORD KEY IS" TO OUT-REC
        MOVE 40 TO CURR
        PERFORM MAS-HYPHEN
        MOVE FM-FIELD TO FIELD
        PERFORM ADD-FIELD.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID01.
    SUBTRACT 1 FROM CURR.
    MOVE "." TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM IO-SKEL.
    MOVE "       FD" TO OUT-REC.
    MOVE 11 TO CURR.
    MOVE 5 TO INDENT.
    MOVE FILE-NAME TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "." TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "       01" TO OUT-REC.
    MOVE 11 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE "RECORD." TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM START-DETAILS.
ID05.
    MOVE "           03" TO OUT-REC.
    MOVE 15 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    IF CURR < 40
         MOVE 40 TO CURR.
    MOVE "PIC" TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-TYPE = "A"
        MOVE "X" TO FIELD
        GO TO ID06.
    IF FM-TYPE = "D"
        MOVE "9(6)" TO FIELD
        PERFORM ADD-FIELD
        GO TO ID10.
    MOVE ZERO TO WORK-G.
    IF FM-TYPE = "I"
```

```
        IF FM-RANGE-I (1) < ZERO
            MOVE "S" TO FIELD
            PERFORM ADD-FIELD
            SUBTRACT 1 FROM CURR.
    IF FM-TYPE = "R"
        IF FM-RANGE-R (1) < ZERO
            MOVE "S" TO FIELD
            PERFORM ADD-FIELD
            SUBTRACT 1 FROM CURR.
    MOVE "9" TO FIELD.
ID06.
    PERFORM ADD-FIELD.
    MOVE "(" TO FIELD.
    SUBTRACT 1 FROM CURR.
     PERFORM ADD-FIELD.
    IF FM-TYPE = "A"
        MOVE FM-SIZE-S TO FIELD
    ELSE
    IF FM-TYPE = "I"
        MOVE FM-SIZE-I TO FIELD
    ELSE
        MOVE FM-SIZE-RB TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
    MOVE ")" TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD
    IF FM-TYPE NOT = "R"
        GO TO ID10.
    SUBTRACT 1 FROM CURR.
    MOVE "V9" TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "(" TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
    MOVE FM-SIZE-RA TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
    MOVE ")" TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
ID10.
    IF FM-OCCURS > 1
        MOVE "OCCURS" TO FIELD
        PERFORM ADD-FIELD
        MOVE FM-OCCURS TO FIELD
        PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "." TO FIELD.
ID11.
    PERFORM ADD-FIELD.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID05.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID20.
    IF FM-KEY-FLAG NOT = 1
        PERFORM READ-DETAILS
```

```
        GO TO ID20.
    MOVE "       01  THIS-KEY            PIC X(20) VALUE"
    TO OUT-REC.
    MOVE 48 TO CURR.
    MOVE FM-DISPLAY-N TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE QUOTE TO ORR (47) ORR (CURR).
    ADD 1 TO CURR.
    MOVE "." TO ORR (CURR).
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    MOVE "            OPEN I-O" TO OUT-REC.
    MOVE 21 TO CURR.
    MOVE FILE-NAME TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    MOVE "." TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
    PERFORM IO-SKEL.
    MOVE "            MOVE" TO OUT-REC.
    MOVE 17 TO CURR.
    MOVE W-QUOTE TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    MOVE FILE-TITLE TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE QUOTE TO ORR (CURR).
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID25.
    IF FM-KEY-FLAG NOT = 1
        PERFORM READ-DETAILS
        GO TO ID25.
    MOVE "            IF THIS-KEY NOT ="
    TO OUT-REC.
    MOVE 30 TO CURR.
    MOVE FM-DISPLAY-N TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE QUOTE TO ORR (29) ORR (CURR).
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    MOVE "            CLOSE" TO OUT-REC.
    MOVE 18 TO CURR.
    MOVE FILE-NAME TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "." TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID30.
    IF FM-KEY-FLAG NOT = ZERO
```

```
        MOVE "            IF THIS-KEY =" TO OUT-REC
        MOVE 27 TO CURR
        MOVE FM-DISPLAY-N TO FIELD
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE QUOTE TO ORR (26) ORR (CURR)
        PERFORM WRITEIT
        MOVE "                PERFORM DISPLAY-STATICS-" TO OUT-REC
        MOVE 40 TO CURR
        MOVE FM-FIELD TO FIELD
        PERFORM ADD-FIELD
        PERFORM WRITEIT
        MOVE "            ELSE" TO OUT-REC
        PERFORM WRITEIT.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID30.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID35.
    IF FM-KEY-FLAG NOT = ZERO
        MOVE "            IF THIS-KEY =" TO OUT-REC
        MOVE 27 TO CURR
        MOVE FM-DISPLAY-N TO FIELD
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE QUOTE TO ORR (26) ORR (CURR)
        PERFORM WRITEIT
        MOVE "                PERFORM INPUT-" TO OUT-REC
        MOVE 30 TO CURR
        MOVE FM-FIELD TO FIELD
        PERFORM ADD-FIELD
        PERFORM WRITEIT
        MOVE "            ELSE" TO OUT-REC
        PERFORM WRITEIT.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID35.
    PERFORM IO-SKEL.
    MOVE "            READ" TO OUT-REC.
    MOVE 5 TO INDENT.
    MOVE 17 TO CURR.
    MOVE FILE-NAME TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "NEXT AT END" TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    MOVE "            PERFORM DISPLAY-" TO OUT-REC.
    MOVE 28 TO CURR.
    MOVE KEY-NAME TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "." TO ORR (CURR).
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID50.
    IF FM-KEY-FLAG NOT = 1
```

```
        PERFORM READ-DETAILS
        GO TO ID50.
    IF FM-KEY-DUPS = "Y"
        MOVE "                GO TO GE99." TO OUT-REC
        PERFORM WRITEIT
    ELSE
        MOVE "                IF NO-RECORD" TO OUT-REC
        PERFORM WRITEIT
        MOVE "                    GO TO GE99" TO OUT-REC
        PERFORM WRITEIT
        MOVE "                ELSE" TO OUT-REC
        PERFORM WRITEIT
        MOVE "                    GO TO GE30." TO OUT-REC
        PERFORM WRITEIT.
    PERFORM IO-SKEL.
    IF FM-KEY-DUPS = "N"
        MOVE "            GO TO GE35." TO OUT-REC
        PERFORM WRITEIT
        MOVE "        GE30." TO OUT-REC
        PERFORM WRITEIT
        MOVE
    "            MOVE  Record already exists  TO ERROR-MESSAGE."
        TO OUT-REC
        MOVE QUOTE TO ORR (17) ORR (39)
        PERFORM WRITEIT
        MOVE "        GE35." TO OUT-REC
        PERFORM WRITEIT.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
    MOVE "            IF INPUT-S =" TO OUT-REC.
    MOVE 25 TO CURR.
ID55.
    IF FM-KEY-FLAG = ZERO
        GO TO ID60.
    IF CURR > 50
        PERFORM WRITEIT
        MOVE 15 TO CURR.
    IF CURR NOT = 25
        MOVE "OR" TO FIELD
        PERFORM ADD-FIELD.
    MOVE QUOTE TO ORR (CURR).
    ADD 1 TO CURR.
    MOVE FM-DISPLAY-N TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE QUOTE TO ORR (CURR).
    ADD 2 TO CURR.
ID60.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID55.
    PERFORM WRITEIT.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
    MOVE "            MOVE  One of:" TO OUT-REC.
    MOVE QUOTE TO ORR (17).
    MOVE 26 TO CURR.
ID65.
    IF FM-KEY-FLAG = ZERO
```

```
        GO TO ID70.
    IF CURR > 50
        SUBTRACT 1 FROM CURR
        MOVE QUOTE TO ORR (CURR)
        ADD 2 TO CURR
        MOVE "TO ERROR-MESSAGE." TO FIELD
        PERFORM ADD-FIELD
        MOVE
        "            CALL  ERRORVDU  USING ROW, COLUMN,"
        TO OUT-REC
        MOVE QUOTE TO ORR (17) ORR (26)
        PERFORM WRITEIT
        MOVE "            ST-SIZE, ERROR-MESSAGE, INPUT-S."
        TO OUT-REC
        PERFORM WRITEIT
        MOVE "            MOVE " TO OUT-REC
        PERFORM WRITEIT
        MOVE QUOTE TO ORR (17)
        MOVE 18 TO CURR.
    IF CURR > 26
        SUBTRACT 1 FROM CURR
        MOVE "," TO ORR (CURR)
        ADD 2 TO CURR.
    MOVE FM-DISPLAY-N TO FIELD.
    PERFORM ADD-FIELD.
ID70.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID65.
    SUBTRACT 1 FROM CURR.
    MOVE QUOTE TO ORR (CURR).
    ADD 2 TO CURR.
    MOVE "TO ERROR-MESSAGE." TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID71.
    IF FM-KEY-FLAG NOT = ZERO
        MOVE "            IF THIS-KEY =" TO OUT-REC
        MOVE 27 TO CURR
        MOVE FM-DISPLAY-N TO FIELD
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE QUOTE TO ORR (26) ORR (CURR)
        PERFORM WRITEIT
        MOVE "                GO TO RV-" TO OUT-REC
        MOVE 25 TO CURR
        MOVE FM-FIELD TO FIELD
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE "." TO ORR (CURR)
        PERFORM WRITEIT.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO ID71.
    PERFORM IO-SKEL.
    PERFORM START-DETAILS.
ID72.
    IF FM-KEY-FLAG NOT = ZERO
```

```
    MOVE "       RV-" TO OUT-REC
    MOVE 11 TO CURR
    MOVE FM-FIELD TO FIELD
    PERFORM ADD-FIELD
    SUBTRACT 1 FROM CURR
    MOVE "." TO ORR (CURR)
    PERFORM WRITEIT
    MOVE "           READ" TO OUT-REC
    MOVE 17 TO CURR
    MOVE FILE-NAME TO FIELD
    PERFORM ADD-FIELD
    MOVE "KEY IS" TO FIELD
    PERFORM ADD-FIELD
    PERFORM MAS-HYPHEN
    MOVE FM-FIELD TO FIELD
    PERFORM ADD-FIELD
    MOVE "INVALID KEY" TO FIELD
    PERFORM ADD-FIELD
    PERFORM WRITEIT
    MOVE "                MOVE 2 TO REPLY." TO OUT-REC
    PERFORM WRITEIT
    MOVE "           GO TO RV99." TO OUT-REC
    PERFORM WRITEIT.
PERFORM READ-DETAILS.
IF NOT END-OF-FILE
    GO TO ID72.
PERFORM IO-SKEL.
PERFORM PERFORM-AUDIT.
MOVE "           WRITE" TO OUT-REC.
MOVE 18 TO CURR.
PERFORM MAS-HYPHEN.
MOVE "RECORD INVALID KEY" TO FIELD.
PERFORM ADD-FIELD.
PERFORM WRITEIT.
PERFORM IO-SKEL.
PERFORM PERFORM-AUDIT.
PERFORM IO-SKEL.
PERFORM PERFORM-AUDIT.
MOVE "           REWRITE" TO OUT-REC.
MOVE 20 TO CURR.
PERFORM MAS-HYPHEN.
MOVE "RECORD INVALID KEY" TO FIELD.
PERFORM ADD-FIELD.
PERFORM WRITEIT.
PERFORM IO-SKEL.
PERFORM PERFORM-AUDIT.
PERFORM IO-SKEL.
PERFORM PERFORM-AUDIT.
PERFORM IO-SKEL.
MOVE "           DELETE" TO OUT-REC.
MOVE 19 TO CURR.
MOVE FILE-NAME TO FIELD.
MOVE 5 TO INDENT.
PERFORM ADD-FIELD.
MOVE "INVALID KEY" TO FIELD.
PERFORM ADD-FIELD.
PERFORM WRITEIT.
PERFORM IO-SKEL.
PERFORM START-DETAILS.
```

```
 ID75.
     IF FM-KEY-FLAG NOT = 1
         PERFORM READ-DETAILS
         GO TO ID75.
     MOVE "                MOVE" TO OUT-REC.
     MOVE 22 TO CURR.
     MOVE FM-DISPLAY-N TO FIELD.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE QUOTE TO ORR (21) ORR (CURR).
     ADD 2 TO CURR.
     MOVE "TO THIS-KEY" TO FIELD.
     PERFORM ADD-FIELD.
     PERFORM WRITEIT.
 ID99.
     EXIT.
*
 PERFORM-AUDIT SECTION.
 PEA00.
     MOVE "            PERFORM AUDIT-" TO OUT-REC.
     MOVE 26 TO CURR.
     MOVE FILE-NAME TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE "." TO FIELD.
     PERFORM ADD-FIELD.
 PEA99.
     EXIT.
*
 XINPUT-FIELDS SECTION.
 XIN00.
     PERFORM IO-SKEL.
     MOVE ZERO TO WORK-B.
     PERFORM START-DETAILS.
 XIN05.
     IF WORK-B = FM-PAGE
         GO TO XIN15.
     IF WORK-B = 0
         GO TO XIN10.
     MOVE "            MOVE 23 TO ROW." TO OUT-REC.
     PERFORM WRITEIT.
     MOVE "            MOVE 1 TO COLUMN." TO OUT-REC.
     PERFORM WRITEIT.
     MOVE "            CALL  CURSORTO  USING ROW, COLUMN."
         TO OUT-REC.
     MOVE QUOTE TO ORR (17) ORR (26).
     PERFORM WRITEIT.
     MOVE "            DISPLAY" TO OUT-REC.
     MOVE 20 TO CURR.
     MOVE W-QUOTE TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE "Next Page" TO FIELD.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE W-QUOTE TO FIELD.
     PERFORM ADD-FIELD.
```

```
    MOVE "WITH NO ADVANCING." TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "           ACCEPT INPUT-S." TO OUT-REC.
    PERFORM WRITEIT.
XIN10.
    MOVE FM-PAGE TO WORK-B.
    MOVE "           MOVE" TO OUT-REC.
    MOVE 17 TO CURR.
    IF WORK-B = ZERO
        MOVE "ZERO" TO FIELD
    ELSE
        MOVE WORK-B TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    MOVE "TO SCREEN-PAGE." TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "           PERFORM NEW-PAGE." TO OUT-REC.
    PERFORM WRITEIT.
XIN15.
    IF FM-KEY-FLAG = 1
        GO TO XIN20.
    IF FM-SCREEN-NU > ZERO
        GO TO XIN16.
    IF FM-OCCURS > 1
        MOVE "           MOVE 1 TO SUBSCRIPT." TO OUT-REC
        PERFORM WRITEIT
        MOVE "       INP-" TO OUT-REC
        MOVE 12 TO CURR
        MOVE 5 TO INDENT
        MOVE FM-FIELD TO FIELD
        PERFORM ADD-FIELD
        MOVE "." TO FIELD
        SUBTRACT 1 FROM CURR
        PERFORM ADD-FIELD.
    MOVE "           MOVE" TO OUT-REC.
    MOVE 17 TO CURR.
    IF FM-TYPE = "A"
        MOVE "SPACES" TO FIELD
    ELSE
        MOVE "ZEROES" TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    MOVE "TO" TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "." TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "           IF SUBSCRIPT <" TO OUT-REC
        MOVE 27 TO CURR
        MOVE FM-OCCURS TO FIELD
        PERFORM ADD-FIELD
        PERFORM WRITEIT
```

```
           MOVE "                ADD 1 TO SUBSCRIPT" TO OUT-REC
           PERFORM WRITEIT
           MOVE "                GO TO INP-" TO OUT-REC
           MOVE 26 TO CURR
           MOVE FM-FIELD TO FIELD
           PERFORM ADD-FIELD
           SUBTRACT 1 FROM CURR
           MOVE "." TO FIELD
           PERFORM ADD-FIELD.
       GO TO XIN20.
 XIN16.
       MOVE "            PERFORM INPUT-" TO OUT-REC.
       MOVE 26 TO CURR.
       MOVE FM-FIELD TO FIELD.
       MOVE 5 TO INDENT.
       PERFORM ADD-FIELD.
       SUBTRACT 1 FROM CURR.
       MOVE "." TO FIELD.
       PERFORM ADD-FIELD.
 XIN20.
       PERFORM READ-DETAILS.
       IF NOT END-OF-FILE
           GO TO XIN05.
 XIN99.
       EXIT.
*
 XAMEND-FIELDS SECTION.
 XAM00.
       PERFORM IO-SKEL.
       MOVE "            MOVE" TO OUT-REC.
       MOVE 17 TO CURR.
       PERFORM START-DETAILS.
 XAM05.
       IF FM-SCREEN-NU > ZERO
           MOVE FM-SCREEN-NU TO FIELD.
       PERFORM READ-DETAILS.
       IF NOT END-OF-FILE
           GO TO XAM05.
       MOVE 5 TO INDENT.
       PERFORM ADD-FIELD.
       MOVE "TO UPPER-I." TO FIELD.
       PERFORM ADD-FIELD.
       PERFORM IO-SKEL.
       MOVE 1 TO WORK-B WORK-A.
       PERFORM START-DETAILS.
 XAM10.
       IF FM-SCREEN-NU > ZERO AND FM-PAGE NOT = WORK-B
           GO TO XAM12.
       IF FM-SCREEN-NU > ZERO
           MOVE FM-SCREEN-NU TO WORK-A.
       PERFORM READ-DETAILS.
       IF END-OF-FILE
           GO TO XAM12.
       GO TO XAM10.
 XAM12.
       MOVE "            IF INPUT-I NOT >" TO OUT-REC.
       MOVE 29 TO CURR.
       MOVE WORK-A TO FIELD.
       MOVE 5 TO INDENT.
```

```
     PERFORM ADD-FIELD.
     PERFORM WRITEIT.
     MOVE "MOVE" TO FIELD.
     MOVE 4 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE WORK-B TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "TO WORK-1" TO FIELD.
     PERFORM ADD-FIELD.
     IF END-OF-FILE
         GO TO XAM15.
     PERFORM WRITEIT.
     MOVE FM-PAGE TO WORK-B.
     MOVE "            ELSE" TO OUT-REC.
     PERFORM WRITEIT.
     GO TO XAM10.
 XAM15.
     SUBTRACT 1 FROM CURR.
     MOVE "." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "            IF WORK-1 NOT = SCREEN-PAGE" TO OUT-REC.
     PERFORM WRITEIT.
     MOVE "                MOVE WORK-1 TO SCREEN-PAGE" TO OUT-REC.
     PERFORM WRITEIT.
     MOVE "                PERFORM SHOW-FIELDS" TO OUT-REC.
     PERFORM WRITEIT.
     MOVE "                GO TO AME00." TO OUT-REC.
     PERFORM WRITEIT.
     PERFORM START-DETAILS.
     IF FM-AMEND = "N" OR FM-KEY-FLAG = 1
*     IF FM-SCREEN-NU = 0
         GO TO XAM25.
 XAM20.
     MOVE "            IF INPUT-I =" TO OUT-REC.
     MOVE 25 TO CURR.
     MOVE FM-SCREEN-NU TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     PERFORM WRITEIT.
     MOVE "PERFORM INPUT-" TO FIELD.
     MOVE 4 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE FM-FIELD TO FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
 XAM25.
     PERFORM READ-DETAILS.
     IF END-OF-FILE
         GO TO XAM30.
     IF FM-SCREEN-NU = 0
         GO TO XAM25.
     PERFORM WRITEIT.
     IF FM-SCREEN-NU > 1
         MOVE "            ELSE" TO OUT-REC
         PERFORM WRITEIT.
     GO TO XAM20.
 XAM30.
```

```
     SUBTRACT 1 FROM CURR.
     MOVE "." TO FIELD.
     PERFORM ADD-FIELD.
 XAM99.
     EXIT.
*
 XSHOW-FIELDS SECTION.
 XSH00.
     PERFORM IO-SKEL.
     MOVE 1 TO WORK-B.
     MOVE "            IF SCREEN-PAGE = 1" TO OUT-REC.
     PERFORM WRITEIT.
     PERFORM START-DETAILS.
 XSH05.
     IF END-OF-FILE
         MOVE "." TO FIELD
         PERFORM ADD-FIELD
         GO TO XSH90.
     IF FM-DISPLAY-N = SPACES OR FM-KEY-FLAG = 1
         PERFORM READ-DETAILS
         GO TO XSH05.
     IF FM-PAGE > WORK-B
         MOVE "." TO FIELD
         PERFORM ADD-FIELD
         MOVE "            IF SCREEN-PAGE =" TO OUT-REC
         MOVE 29 TO CURR
         MOVE FM-PAGE TO FIELD WORK-B
         MOVE 5 TO INDENT
         PERFORM ADD-FIELD.
     PERFORM WRITEIT.
     MOVE "                PERFORM DISPLAY-" TO OUT-REC.
     MOVE 32 TO CURR.
     MOVE FM-FIELD TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     PERFORM READ-DETAILS.
     GO TO XSH05.
 XSH90.
     PERFORM IO-SKEL.
     MOVE "            OR SCREEN-PAGE =" TO OUT-REC.
     MOVE 29 TO CURR.
     MOVE WORK-B TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     PERFORM WRITEIT.
 XSH99.
     EXIT.
*
 XAUDIT-MASTER SECTION.
 XAU00.
     PERFORM IO-SKEL.
     MOVE "       AUDIT-" TO OUT-REC.
     MOVE 14 TO CURR.
     MOVE FILE-NAME TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "SECTION." TO FIELD.
     PERFORM ADD-FIELD.
     PERFORM IO-SKEL.
```

```
    PERFORM START-DETAILS.
XAU05.
    MOVE "            MOVE" TO OUT-REC.
    MOVE 17 TO CURR.
    MOVE W-QUOTE TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    IF FM-DISPLAY-N = SPACES
        MOVE FM-FIELD TO FIELD
    ELSE
        MOVE FM-DISPLAY-N TO FIELD.
    PERFORM ADD-FIELD.
    MOVE W-QUOTE TO FIELD.
    SUBTRACT 1 FROM CURR.
    PERFORM ADD-FIELD.
    MOVE "TO AUDIT-NAME." TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "            MOVE 1 TO SUBSCRIPT." TO OUT-REC
        PERFORM WRITEIT
        MOVE "AUDIT-" TO FIELD
        MOVE 2 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE FM-FIELD TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE "." TO FIELD
        PERFORM ADD-FIELD.
    IF FM-TYPE = "A"
        GO TO XAU06.
    MOVE "            IF" TO OUT-REC.
    MOVE 15 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "NUMERIC" TO FIELD.
    PERFORM ADD-FIELD
    PERFORM WRITEIT.
    MOVE "                MOVE" TO OUT-REC.
    MOVE 21 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "TO AUDIT-VALUE-" TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    IF FM-TYPE NOT = "R"
        MOVE "9" TO FIELD
```

```
    ELSE
        MOVE "R" TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM WRITEIT.
    MOVE "            ELSE" TO OUT-REC.
    PERFORM WRITEIT.
    IF FM-TYPE = "R"
        MOVE
        "                MOVE  contents undefined" TO OUT-REC
        MOVE QUOTE TO ORR (21) ORR (40)
        MOVE 42 TO CURR
        GO TO XAU05A.
    MOVE "                MOVE" TO OUT-REC.
    MOVE 21 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
XAU05A.
    MOVE "TO AUDIT-VALUE-X." TO FIELD.
    PERFORM ADD-FIELD.
    GO TO XAU07.
XAU06.
    MOVE "            MOVE" TO OUT-REC.
    MOVE 17 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "TO AUDIT-VALUE-X." TO FIELD.
    PERFORM ADD-FIELD.
XAU07.
    MOVE "            PERFORM AUDIT-PRINT." TO OUT-REC.
    PERFORM WRITEIT.
    IF FM-OCCURS > 1
        MOVE "            IF SUBSCRIPT NOT =" TO OUT-REC
        MOVE 31 TO CURR
        MOVE FM-OCCURS TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        PERFORM WRITEIT
        MOVE "                ADD 1 TO SUBSCRIPT" TO OUT-REC
        PERFORM WRITEIT
        MOVE "GO TO AUDIT-" TO FIELD
        MOVE 4 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE FM-FIELD TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        MOVE "." TO FIELD
        PERFORM ADD-FIELD.
XAU10.
```

```
     PERFORM READ-DETAILS.
     IF NOT END-OF-FILE
         GO TO XAU05.
     PERFORM IO-SKEL.
 XAU99.
     EXIT.
*
 COMMENT-LINE SECTION.
 COL00.
     MOVE "       *" TO OUT-REC.
     PERFORM WRITEIT.
 COL99.
     EXIT.
*
 XDISPLAY-FIELD SECTION.
 XDI00.
     PERFORM IO-SKEL.
     PERFORM START-DETAILS.
 XDI05.
     IF FM-DISPLAY-N = SPACES
         GO TO XDI20.
     PERFORM COMMENT-LINE.
     MOVE "DISPLAY-" TO FIELD.
     MOVE 2 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE FM-FIELD TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "SECTION." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "D-" TO FIELD.
     MOVE 2 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE FM-FIELD TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE "-00." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "           MOVE" TO OUT-REC.
     MOVE 17 TO CURR.
     MOVE FM-LINE TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "TO ROW." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "           MOVE 30 TO COLUMN." TO OUT-REC.
     PERFORM WRITEIT.
     IF FM-OCCURS > 1
         MOVE "           MOVE 1 TO SUBSCRIPT." TO OUT-REC
         PERFORM WRITEIT
         MOVE "D-" TO FIELD
         MOVE 2 TO INDENT
         PERFORM ADD-FIELD
         SUBTRACT 1 FROM CURR
         MOVE FM-FIELD TO FIELD
         MOVE 5 TO INDENT
```

```
    PERFORM ADD-FIELD
    SUBTRACT 1 FROM CURR
    MOVE "-05." TO FIELD
    PERFORM ADD-FIELD.
IF FM-TYPE = "A"
    GO TO XDI10.
MOVE "           IF" TO OUT-REC.
MOVE 15 TO CURR.
MOVE 5 TO INDENT.
PERFORM MAS-HYPHEN.
MOVE FM-FIELD TO FIELD.
PERFORM ADD-FIELD.
IF FM-OCCURS > 1
    MOVE "(SUBSCRIPT)" TO FIELD
    PERFORM ADD-FIELD.
MOVE "NUMERIC" TO FIELD.
PERFORM ADD-FIELD.
PERFORM WRITEIT.
MOVE "                MOVE" TO OUT-REC.
MOVE 21 TO CURR.
PERFORM MAS-HYPHEN.
MOVE FM-FIELD TO FIELD.
MOVE 5 TO INDENT.
PERFORM ADD-FIELD.
IF FM-OCCURS > 1
    MOVE "(SUBSCRIPT)" TO FIELD
    PERFORM ADD-FIELD.
MOVE "TO OUTPUT-" TO FIELD.
PERFORM ADD-FIELD.
SUBTRACT 1 FROM CURR.
IF FM-TYPE NOT = "R"
    MOVE "I" TO FIELD
ELSE
    MOVE "R" TO FIELD.
PERFORM ADD-FIELD.
PERFORM WRITEIT.
MOVE "                CALL  OUTPUT" TO OUT-REC.
MOVE 28 TO CURR.
IF FM-TYPE NOT = "R"
    MOVE "I" TO ORR (CURR)
ELSE
    MOVE "R" TO ORR (CURR).
MOVE QUOTE TO ORR (21), ORR (29).
MOVE 5 TO INDENT.
MOVE 31 TO CURR.
MOVE "USING ROW, COLUMN, OUTPUT-" TO FIELD.
PERFORM ADD-FIELD.
SUBTRACT 1 FROM CURR.
IF FM-TYPE NOT = "R"
    MOVE "I" TO ORR (CURR)
ELSE
    MOVE "R" TO ORR (CURR).
PERFORM WRITEIT.
MOVE "           ELSE" TO OUT-REC.
PERFORM WRITEIT.
MOVE
"               CALL  CURSORTO  USING ROW, COLUMN"
    TO OUT-REC.
MOVE QUOTE TO ORR (21) ORR (30).
```

```
    PERFORM WRITEIT.
    MOVE "                DISPLAY" TO OUT-REC.
    MOVE 24 TO CURR.
    MOVE 5 TO INDENT.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "WITH NO ADVANCING." TO FIELD.
    PERFORM ADD-FIELD.
    GO TO XDI15.
XDI10.
    MOVE
    "           CALL  CURSORTO  USING ROW, COLUMN."
        TO OUT-REC.
    MOVE QUOTE TO ORR (17) ORR (26).
    PERFORM WRITEIT.
    MOVE "           DISPLAY" TO OUT-REC.
    MOVE 20 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "WITH NO ADVANCING." TO FIELD.
    PERFORM ADD-FIELD.
XDI15.
    IF FM-OCCURS > 1
        MOVE "           IF SUBSCRIPT <" TO OUT-REC
        MOVE 27 TO CURR
        MOVE FM-OCCURS TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        PERFORM WRITEIT
        MOVE "               ADD 1 TO SUBSCRIPT" TO OUT-REC
        PERFORM WRITEIT
        MOVE "               ADD 1 TO ROW" TO OUT-REC
        PERFORM WRITEIT
        MOVE "GO TO D-" TO FIELD
        MOVE 4 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE FM-FIELD TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE "-05." TO FIELD
        PERFORM ADD-FIELD.
    MOVE "D-" TO FIELD.
    MOVE 2 TO INDENT.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
```

```
     SUBTRACT 1 FROM CURR.
     MOVE "-99." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "            EXIT." TO OUT-REC.
     PERFORM WRITEIT.
 XDI20.
     PERFORM READ-DETAILS.
     IF NOT END-OF-FILE
         GO TO XDI05.
 XDI99.
     EXIT.
*
 XINPUT-FIELD SECTION.
 XINP00.
     PERFORM IO-SKEL.
     PERFORM START-DETAILS.
 XINP05.
     IF FM-AMEND = "N"
         GO TO XINP40.
     PERFORM COMMENT-LINE.
     MOVE "INPUT-" TO FIELD.
     MOVE 2 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE FM-FIELD TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "SECTION." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "I-" TO FIELD.
     MOVE 2 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE FM-FIELD TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE "-00." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "            MOVE" TO OUT-REC.
     MOVE 17 TO CURR.
     MOVE FM-LINE TO FIELD.
     MOVE 5 TO INDENT.
     PERFORM ADD-FIELD.
     MOVE "TO ROW." TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "            MOVE 30 TO COLUMN." TO OUT-REC.
     PERFORM WRITEIT.
     IF FM-TYPE = "A"
         GO TO XINP10.
     MOVE "            MOVE" TO OUT-REC.
     MOVE 17 TO CURR.
     IF FM-TYPE = "I"
         IF FM-RANGE-I (1) = ZERO
             MOVE "ZERO" TO FIELD
         ELSE
             MOVE FM-RANGE-I (1) TO ST-INTEGER
             MOVE ST-INTEGER TO FIELD
     ELSE
```

```
IF FM-TYPE = "D"
    IF FM-RANGE-D (1) = ZERO
        MOVE "ZERO" TO FIELD
    ELSE
        MOVE FM-RANGE-D (1) TO FIELD
ELSE
    IF FM-RANGE-R (1) = ZERO
        MOVE "ZERO" TO FIELD
    ELSE
        MOVE FM-RANGE-R (1) TO ST-REAL
        PERFORM BLANK-REAL
        MOVE ST-REAL TO FIELD.
MOVE 5 TO INDENT.
PERFORM ADD-FIELD.
MOVE "TO LOWER-" TO FIELD.
PERFORM ADD-FIELD.
SUBTRACT 1 FROM CURR.
IF FM-TYPE = "D"
    MOVE "D" TO FIELD
ELSE
IF FM-TYPE = "I"
    MOVE "I" TO FIELD
ELSE
    MOVE "R" TO FIELD.
PERFORM ADD-FIELD.
MOVE "." TO FIELD.
PERFORM ADD-FIELD.
MOVE "           MOVE" TO OUT-REC.
MOVE 17 TO CURR.
IF FM-TYPE = "I"
    IF FM-RANGE-I (2) = ZERO
        MOVE "ZERO" TO FIELD
    ELSE
        MOVE FM-RANGE-I (2) TO ST-INTEGER
        MOVE ST-INTEGER TO FIELD
ELSE
IF FM-TYPE = "D"
    IF FM-RANGE-D (2) = ZERO
        MOVE "ZERO" TO FIELD
    ELSE
        MOVE FM-RANGE-D (2) TO FIELD
ELSE
    IF FM-RANGE-R (2) = ZERO
        MOVE "ZERO" TO FIELD
    ELSE
        MOVE FM-RANGE-R (2) TO ST-REAL
        PERFORM BLANK-REAL
        MOVE ST-REAL TO FIELD.
MOVE 5 TO INDENT.
PERFORM ADD-FIELD.
MOVE "TO UPPER-" TO FIELD.
PERFORM ADD-FIELD.
SUBTRACT 1 FROM CURR.
IF FM-TYPE = "D"
    MOVE "D" TO FIELD
ELSE
IF FM-TYPE = "I"
    MOVE "I" TO FIELD
ELSE
```

```
            MOVE "R" TO FIELD.
        PERFORM ADD-FIELD.
        MOVE "." TO FIELD.
        PERFORM ADD-FIELD.
        IF FM-TYPE = "R"
            MOVE "            MOVE" TO OUT-REC
            MOVE 17 TO CURR
            MOVE FM-SIZE-RA TO FIELD
            MOVE 5 TO INDENT
            PERFORM ADD-FIELD
            MOVE "TO DECIMALS." TO FIELD
            PERFORM ADD-FIELD.
XINP10.
        IF FM-OCCURS > 1
            MOVE "            MOVE 1 TO SUBSCRIPT." TO OUT-REC
            PERFORM WRITEIT.
        MOVE "I-" TO FIELD.
        MOVE 2 TO INDENT.
        PERFORM ADD-FIELD.
        SUBTRACT 1 FROM CURR.
        MOVE FM-FIELD TO FIELD.
        MOVE 5 TO INDENT.
        PERFORM ADD-FIELD.
        SUBTRACT 1 FROM CURR.
        MOVE "-05." TO FIELD.
        PERFORM ADD-FIELD.
        IF FM-TYPE = "A"
            MOVE "            MOVE" TO OUT-REC
            MOVE 17 TO CURR
            MOVE FM-SIZE-S TO FIELD
            MOVE 5 TO INDENT
            PERFORM ADD-FIELD
            MOVE "TO ST-SIZE." TO FIELD
            PERFORM ADD-FIELD
            MOVE "            CALL  INPUTS  USING ROW, COLUMN,"
            TO OUT-REC
            MOVE QUOTE TO ORR (17) ORR (24)
            PERFORM WRITEIT
            MOVE "            INPUT-S, ST-SIZE." TO OUT-REC
            PERFORM WRITEIT
        ELSE
        IF FM-KEY-FLAG = ZERO
          IF FM-TYPE = "D"
            MOVE "            CALL  INPUTD  USING ROW, COLUMN,"
            TO OUT-REC
            MOVE QUOTE TO ORR (17) ORR (24)
            PERFORM WRITEIT
            MOVE "            LOWER-D, UPPER-D, INPUT-D."
            TO OUT-REC
            PERFORM WRITEIT
          ELSE
          IF FM-TYPE = "I"
            MOVE "            CALL  INPUTI  USING ROW, COLUMN,"
            TO OUT-REC
            MOVE QUOTE TO ORR (17) ORR (24)
            PERFORM WRITEIT
            MOVE "            LOWER-I, UPPER-I, INPUT-I."
            TO OUT-REC
            PERFORM WRITEIT
```

```
   ELSE
     MOVE "            CALL  INPUTR  USING ROW, COLUMN,"
     TO OUT-REC
     MOVE QUOTE TO ORR (17) ORR (24)
     PERFORM WRITEIT
     MOVE "            LOWER-R, UPPER-R, INPUT-R,"
     TO OUT-REC
     PERFORM WRITEIT
     MOVE "            DECIMALS." TO OUT-REC
     PERFORM WRITEIT
ELSE
   IF FM-TYPE = "D"
     MOVE "            CALL  INPUTDK  USING ROW, COLUMN,"
     TO OUT-REC
     MOVE QUOTE TO ORR (17) ORR (25)
     PERFORM WRITEIT
     MOVE "            LOWER-D, UPPER-D, INPUT-D, INPUT-S."
     TO OUT-REC
     PERFORM WRITEIT
   ELSE
   IF FM-TYPE = "I"
     MOVE "            CALL  INPUTIK  USING ROW, COLUMN,"
     TO OUT-REC
     MOVE QUOTE TO ORR (17) ORR (25)
     PERFORM WRITEIT
     MOVE "            LOWER-I, UPPER-I, INPUT-I, INPUT-S."
     TO OUT-REC
     PERFORM WRITEIT
   ELSE
     MOVE "            CALL  INPUTRK  USING ROW, COLUMN,"
     TO OUT-REC
     MOVE QUOTE TO ORR (17) ORR (25)
     PERFORM WRITEIT
     MOVE "            LOWER-R, UPPER-R, INPUT-R,"
     TO OUT-REC
     PERFORM WRITEIT
     MOVE "            INPUT-S, DECIMALS." TO OUT-REC
     PERFORM WRITEIT.
MOVE 5 TO INDENT.
IF FM-KEY-FLAG > ZERO
     MOVE "            IF THIS-KEY =" TO OUT-REC
     MOVE 27 TO CURR
     MOVE FM-DISPLAY-N TO FIELD
     PERFORM ADD-FIELD
     SUBTRACT 1 FROM CURR
     MOVE QUOTE TO ORR (26) ORR (CURR)
     PERFORM WRITEIT
     MOVE "            AND (INPUT-S =  +  OR SPACES)"
     TO OUT-REC
     MOVE QUOTE TO ORR (27) ORR (29)
     PERFORM WRITEIT
     MOVE "                GO TO I-" TO OUT-REC
     MOVE 24 TO CURR
     MOVE FM-FIELD TO FIELD
     PERFORM ADD-FIELD
     SUBTRACT 1 FROM CURR
     MOVE "-99." TO FIELD
     PERFORM ADD-FIELD.
MOVE "          MOVE INPUT-" TO OUT-REC.
```

```
    MOVE 23 TO CURR.
    IF FM-TYPE = "A"
        MOVE "S" TO FIELD
    ELSE
    IF FM-TYPE = "D"
        MOVE "D" TO FIELD
    ELSE
    IF FM-TYPE = "I"
        MOVE "I" TO FIELD
    ELSE
        MOVE "R" TO FIELD.
    MOVE 5 TO INDENT.
XINP14.
    PERFORM ADD-FIELD.
    MOVE "TO" TO FIELD.
    PERFORM ADD-FIELD.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "." TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-NUM-VALS = ZERO
        GO TO XINP35.
    MOVE "           IF" TO OUT-REC.
    MOVE 15 TO CURR.
    PERFORM MAS-HYPHEN.
    MOVE FM-FIELD TO FIELD.
    PERFORM ADD-FIELD.
    IF FM-OCCURS > 1
        MOVE "(SUBSCRIPT)" TO FIELD
        PERFORM ADD-FIELD.
    MOVE "=" TO FIELD.
    PERFORM ADD-FIELD.
    MOVE 1 TO WORK-B.
XINP15.
    IF FM-TYPE = "I"
    OR FM-TYPE = "R"
    OR FM-TYPE = "D"
    OR FM-VALUES-S (WORK-B) = SPACES
        GO TO XINP25.
    MOVE W-QUOTE TO FIELD.
    MOVE FM-VALUES-S (WORK-B) TO ST-VALUES.
    MOVE 1 TO WORK-C.
    MOVE 2 TO WORK-D.
XINP20.
    IF VIN (WORK-C) < " " OR VIN (WORK-C) > "~"
        GO TO XINP21.
    MOVE VIN (WORK-C) TO FI (WORK-D).
XINP21.
    ADD 1 TO WORK-C.
    ADD 1 TO WORK-D.
    IF VIN (WORK-C) NOT = SPACE
        GO TO XINP20.
    MOVE W-QUOTE TO FI (WORK-D).
    GO TO XINP30.
XINP25.
```

```
        IF FM-TYPE = "A"
            IF FM-VALUES-S (WORK-B) = SPACES
                MOVE "SPACES" TO FIELD
            ELSE
                MOVE FM-VALUES-S (WORK-B) TO FIELD
        ELSE
        IF FM-TYPE = "I"
            IF FM-VALUES-I (WORK-B) = ZERO
                MOVE "ZERO" TO FIELD
            ELSE
                MOVE FM-VALUES-I (WORK-B) TO ST-INTEGER
                MOVE ST-INTEGER TO FIELD
        ELSE
        IF FM-TYPE = "R"
            IF FM-VALUES-R (WORK-B) = ZERO
                MOVE "ZERO" TO FIELD
            ELSE
                MOVE FM-VALUES-R (WORK-B) TO ST-REAL
                PERFORM BLANK-REAL
                MOVE ST-REAL TO FIELD
        ELSE
        IF FM-TYPE = "D"
            IF FM-VALUES-D (WORK-B) = ZERO
                MOVE "ZERO" TO FIELD
            ELSE
                MOVE FM-VALUES-D (WORK-B) TO FIELD.
    XINP30.
        PERFORM ADD-FIELD.
        IF FM-NUM-VALS > WORK-B
            ADD 1 TO WORK-B
            MOVE "OR" TO FIELD
            PERFORM ADD-FIELD
            GO TO XINP15.
        PERFORM WRITEIT.
        MOVE "GO TO I-" TO FIELD.
        MOVE 4 TO INDENT.
        PERFORM ADD-FIELD.
        SUBTRACT 1 FROM CURR.
        MOVE FM-FIELD TO FIELD.
        MOVE 5 TO INDENT.
        PERFORM ADD-FIELD.
        SUBTRACT 1 FROM CURR.
        IF FM-OCCURS > 1
            MOVE "-10." TO FIELD
        ELSE
            MOVE "-99." TO FIELD.
        PERFORM ADD-FIELD.
    *
    * build up the value error message lines
    *
        MOVE "           MOVE" TO OUT-REC.
        MOVE QUOTE TO ORR (17).
        MOVE "One of" TO FIELD.
        MOVE 5 TO INDENT.
        MOVE 18 TO CURR.
        PERFORM ADD-FIELD.
        ADD 1 TO CURR.
        MOVE 1 TO WORK-B.
    XINP31.
```

```
IF FM-NUM-VALS < WORK-B
    MOVE QUOTE TO ORR (CURR)
    ADD 2 TO CURR
    MOVE "TO ERROR-MESSAGE." TO FIELD
    MOVE 5 TO INDENT
    PERFORM ADD-FIELD
    MOVE
    "           CALL  ERRORVDU  USING ROW, COLUMN,"
    TO OUT-REC
    MOVE QUOTE TO ORR (17) ORR (26)
    PERFORM WRITEIT
    MOVE "            ST-SIZE, ERROR-MESSAGE, INPUT-S."
    TO OUT-REC
    PERFORM WRITEIT
    GO TO XINP32.
IF FM-TYPE = "A"
    IF FM-VALUES-S (WORK-B) = SPACES
         MOVE "or spaces" TO FIELD
         PERFORM ADD-FIELD
         ADD 1 TO WORK-B
         GO TO XINP31.
IF FM-TYPE = "I"
    IF FM-VALUES-I (WORK-B) = ZERO
         MOVE "or zero" TO FIELD
         PERFORM ADD-FIELD
         ADD 1 TO WORK-B
         GO TO XINP31.
IF FM-TYPE = "D"
    IF FM-VALUES-D (WORK-B) = ZERO
         MOVE "or zero" TO FIELD
         PERFORM ADD-FIELD
         ADD 1 TO WORK-B
         GO TO XINP31.
IF FM-TYPE = "R"
    IF FM-VALUES-R (WORK-B) = ZERO
         MOVE "or zero" TO FIELD
         PERFORM ADD-FIELD
         ADD 1 TO WORK-B
         GO TO XINP31.
PERFORM TEST-LINE.
IF WORK-B > 1
    MOVE "or" TO FIELD
    PERFORM ADD-FIELD.
IF FM-TYPE = "A"
    MOVE FM-VALUES-S (WORK-B) TO FIELD
ELSE
IF FM-TYPE = "I"
    MOVE FM-VALUES-I (WORK-B) TO ST-INTEGER
    MOVE ST-INTEGER TO FIELD
ELSE
IF FM-TYPE = "D"
    MOVE FM-VALUES-D (WORK-B) TO FIELD
ELSE
    MOVE FM-VALUES-R (WORK-B) TO ST-REAL
    PERFORM BLANK-REAL
    MOVE ST-REAL TO FIELD.
PERFORM ADD-FIELD.
ADD 1 TO WORK-B.
GO TO XINP31.
```

```
XINP32.
    MOVE "            GO TO I-" TO OUT-REC.
    MOVE 20 TO CURR.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "-05." TO FIELD.
    PERFORM ADD-FIELD.
XINP35.
    IF FM-OCCURS > 1
        MOVE "I-" TO FIELD
        MOVE 2 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE FM-FIELD TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE "-10." TO FIELD
        PERFORM ADD-FIELD
        MOVE "            IF SUBSCRIPT <" TO OUT-REC
        MOVE 27 TO CURR
        MOVE FM-OCCURS TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        PERFORM WRITEIT
        MOVE "                ADD 1 TO SUBSCRIPT" TO OUT-REC
        PERFORM WRITEIT
        MOVE "                ADD 1 TO ROW" TO OUT-REC
        PERFORM WRITEIT
        MOVE "GO TO I-" TO FIELD
        MOVE 4 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE FM-FIELD TO FIELD
        MOVE 5 TO INDENT
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE "-05." TO FIELD
        PERFORM ADD-FIELD.
    MOVE "I-" TO FIELD.
    MOVE 2 TO INDENT.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE FM-FIELD TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "-99." TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "            EXIT." TO OUT-REC.
    PERFORM WRITEIT.
XINP40.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
       GO TO XINP05.
XINP99.
    EXIT.
```

```
*
 BLANK-REAL SECTION.
 BL00.
     MOVE 19 TO WORK-K.
 BL05.
     IF STR (WORK-K) NOT = "0"
         GO TO BL99.
     MOVE SPACE TO STR (WORK-K).
     IF WORK-K > 16
         SUBTRACT 1 FROM WORK-K
         GO TO BL05.
 BL99.
     EXIT.
*
*
 TEST-LINE SECTION.
 TE00.
     IF CURR < 50
         GO TO TE99.
     MOVE "o" TO ORR (70).
     MOVE "r" TO ORR (71).
     MOVE QUOTE TO ORR (72).
     PERFORM WRITEIT.
     MOVE "            TO ERROR-MESSAGE." TO OUT-REC.
     PERFORM WRITEIT.
     MOVE
     "            CALL ERRORVDU  USING ROW, COLUMN,"
     TO OUT-REC.
     MOVE QUOTE TO ORR (17) ORR (26).
     PERFORM WRITEIT.
     MOVE "            ST-SIZE, ERROR-MESSAGE, INPUT-S."
     TO OUT-REC.
     PERFORM WRITEIT.
     MOVE "            MOVE" TO OUT-REC.
     MOVE QUOTE TO ORR (17).
     MOVE 18 TO CURR.
 TE99.
     EXIT.
*
 XNEW-PAGE SECTION.
 XNP00.
     PERFORM IO-SKEL.
     PERFORM START-DETAILS.
     MOVE 1 TO WORK-B.
     MOVE "            IF SCREEN-PAGE = 1" TO OUT-REC
     PERFORM WRITEIT.
 XNP05.
     IF END-OF-FILE
         MOVE "." TO FIELD
         MOVE 5 TO INDENT
         PERFORM ADD-FIELD
         GO TO XNP99.
     IF FM-KEY-FLAG = 1 OR FM-DISPLAY-N = SPACES
         PERFORM READ-DETAILS
         GO TO XNP05.
     IF WORK-B < FM-PAGE
         MOVE "." TO FIELD
         MOVE 5 TO INDENT
         PERFORM ADD-FIELD
```

```
           MOVE FM-PAGE TO WORK-B
           MOVE "            IF SCREEN-PAGE =" TO OUT-REC
           MOVE 29 TO CURR
           MOVE WORK-B TO FIELD
           MOVE 5 TO INDENT
           PERFORM ADD-FIELD.
       PERFORM WRITEIT.
       MOVE "                PERFORM DISPLAY-STATICS-" TO OUT-REC.
       MOVE 40 TO CURR.
       MOVE 5 TO INDENT.
       MOVE FM-FIELD TO FIELD.
       PERFORM ADD-FIELD.
       PERFORM READ-DETAILS.
       GO TO XNP05.
   XNP99.
       EXIT.
  *
   DISPLAY-STATICS SECTION.
   DIS00.
       PERFORM IO-SKEL.
       PERFORM START-DETAILS.
   DIS05.
       IF FM-DISPLAY-N = SPACES
           GO TO DIS10.
       PERFORM COMMENT-LINE.
       MOVE "       DISPLAY-STATICS-" TO OUT-REC.
       MOVE 24 TO CURR.
       MOVE 5 TO INDENT.
       MOVE FM-FIELD TO FIELD.
       PERFORM ADD-FIELD.
       MOVE "SECTION." TO FIELD.
       PERFORM ADD-FIELD.
       MOVE "       DS-" TO OUT-REC.
       MOVE 11 TO CURR.
       MOVE 5 TO INDENT.
       MOVE FM-FIELD TO FIELD.
       PERFORM ADD-FIELD.
       SUBTRACT 1 FROM CURR.
       MOVE "-00." TO FIELD.
       PERFORM ADD-FIELD.
       MOVE "           MOVE" TO OUT-REC.
       MOVE 17 TO CURR.
       MOVE FM-LINE TO FIELD.
       MOVE 5 TO INDENT.
       PERFORM ADD-FIELD.
       MOVE "TO ROW." TO FIELD.
       PERFORM ADD-FIELD.
       IF FM-SCREEN-NU = ZERO
           GO TO DIS06.
       MOVE "           MOVE 1 TO COLUMN." TO OUT-REC.
       PERFORM WRITEIT.
       MOVE
       "           CALL  CURSORTO  USING ROW, COLUMN."
           TO OUT-REC.
       MOVE QUOTE TO ORR (17) ORR (26).
       PERFORM WRITEIT.
       MOVE "           DISPLAY" TO OUT-REC.
       MOVE 20 TO CURR.
       MOVE 5 TO INDENT.
```

```
     MOVE FM-SCREEN-NU TO FIELD.
     PERFORM ADD-FIELD.
     MOVE "WITH NO ADVANCING." TO FIELD.
     PERFORM ADD-FIELD.
 DIS06.
     MOVE "            MOVE 3 TO COLUMN." TO OUT-REC.
     PERFORM WRITEIT.
     MOVE
     "            CALL  CURSORTO  USING ROW, COLUMN."
         TO OUT-REC.
     MOVE QUOTE TO ORR (17) ORR (26).
     PERFORM WRITEIT.
     MOVE "            DISPLAY" TO OUT-REC.
     MOVE QUOTE TO ORR (20).
*     IF FM-SCREEN-NU > ZERO
*         MOVE ":" TO ORR(21).
     MOVE 22 TO CURR.
     MOVE 5 TO INDENT.
     MOVE FM-DISPLAY-N TO FIELD.
     PERFORM ADD-FIELD.
     SUBTRACT 1 FROM CURR.
     MOVE QUOTE TO ORR (CURR).
     ADD 2 TO CURR.
     MOVE "WITH NO ADVANCING." TO FIELD.
     PERFORM ADD-FIELD.
     IF FM-OCCURS > 1
         MOVE "            MOVE 1 TO SUBSCRIPT." TO OUT-REC
         PERFORM WRITEIT
         MOVE "        DS-" TO OUT-REC
         MOVE 11 TO CURR
         MOVE 5 TO INDENT
         MOVE FM-FIELD TO FIELD
         PERFORM ADD-FIELD
         SUBTRACT 1 FROM CURR
         MOVE "-05." TO FIELD
         PERFORM ADD-FIELD.
     IF FM-SCREEN-NU = ZERO AND FM-KEY-FLAG NOT = 1
         GO TO DIS07.
     MOVE "            MOVE 29 TO COLUMN." TO OUT-REC.
     PERFORM WRITEIT.
     MOVE
     "            CALL  CURSORTO  USING ROW, COLUMN."
         TO OUT-REC.
     MOVE QUOTE TO ORR (17) ORR (26).
     PERFORM WRITEIT.
     MOVE "            DISPLAY  :  WITH NO ADVANCING."
     TO OUT-REC.
     MOVE QUOTE TO ORR (20) ORR (22).
     PERFORM WRITEIT.
     IF FM-TYPE = "A"
         ADD 30 FM-SIZE-S GIVING WORK-A
     ELSE
     IF FM-TYPE = "D"
         MOVE 36 TO WORK-A
     ELSE
     IF FM-TYPE = "I"
         ADD 30 FM-SIZE-I GIVING WORK-A
     ELSE
     IF FM-TYPE = "R"
```

```
        ADD 30 FM-SIZE-RB FM-SIZE-RA GIVING WORK-A.
    MOVE ZERO TO WORK-B.
    IF FM-TYPE = "I" AND FM-RANGE-I (1) < ZERO
        ADD 1 TO WORK-A.
    IF FM-TYPE = "R" AND FM-RANGE-R (1) < ZERO
        ADD 1 TO WORK-A.
    IF FM-TYPE = "R"
        ADD 1 TO WORK-A.
    IF WORK-A > 80
        GO TO DIS07.
    MOVE "           MOVE" TO OUT-REC.
    MOVE 17 TO CURR.
    MOVE WORK-A TO FIELD.
    MOVE 5 TO INDENT.
    PERFORM ADD-FIELD.
    MOVE "TO COLUMN." TO FIELD.
    PERFORM ADD-FIELD.
    MOVE
    "           CALL  CURSORTO  USING ROW, COLUMN."
        TO OUT-REC.
    MOVE QUOTE TO ORR (17) ORR (26).
    PERFORM WRITEIT.
    MOVE "           DISPLAY  :  WITH NO ADVANCING."
    TO OUT-REC.
    MOVE QUOTE TO ORR (20) ORR (22).
    PERFORM WRITEIT.
DIS07.
    IF FM-OCCURS > 1
        MOVE "           IF SUBSCRIPT <" TO OUT-REC
        MOVE 26 TO CURR
        MOVE 5 TO INDENT
        MOVE FM-OCCURS TO FIELD
        PERFORM ADD-FIELD
        PERFORM WRITEIT
        MOVE "               ADD 1 TO ROW"
            TO OUT-REC
        PERFORM WRITEIT
        MOVE "               ADD 1 TO SUBSCRIPT"
            TO OUT-REC
        PERFORM WRITEIT
        MOVE "               GO TO DS-" TO OUT-REC
        MOVE 25 TO CURR
        MOVE 5 TO INDENT
        MOVE FM-FIELD TO FIELD
        PERFORM ADD-FIELD
        SUBTRACT 1 FROM CURR
        MOVE "-05." TO FIELD
        PERFORM ADD-FIELD.
    MOVE "       DS-" TO OUT-REC.
    MOVE 11 TO CURR.
    MOVE 5 TO INDENT.
    MOVE FM-FIELD TO FIELD.
    PERFORM ADD-FIELD.
    SUBTRACT 1 FROM CURR.
    MOVE "-99." TO FIELD.
    PERFORM ADD-FIELD.
    MOVE "           EXIT." TO OUT-REC.
    PERFORM WRITEIT.
DIS10.
```

```
     PERFORM READ-DETAILS.
     IF NOT END-OF-FILE
         GO TO DIS05.
 DIS99.
     EXIT.
*
 ADD-FIELD SECTION.
 AF00.
     IF FIELD = SPACES
        GO TO AF99.
     IF NO-INDENT
        GO TO AF05.
     IF NOT SAME-LINE AND OUT-REC NOT = SPACES
         PERFORM WRITEIT.
     IF INDENT = 1
         MOVE "*" TO ORR (7)
         ADD 2 TO CURR
         GO TO AF05.
     IF INDENT NOT = 1 AND CURR = 7
         MOVE  8 TO CURR.
     IF AREA-A
         MOVE 8 TO CURR.
     IF AREA-B
         MOVE 12 TO CURR.
     IF INDENTED
         MOVE 16 TO CURR.
 AF05.
     MOVE 60 TO WORK-F.
 AF10.
     IF FI (WORK-F) = SPACE
         SUBTRACT 1 FROM WORK-F
         GO TO AF10.
     IF NOT SAME-LINE
         GO TO AF15.
     ADD WORK-F CURR GIVING WORK-E.
     IF WORK-E > 72
         PERFORM WRITEIT
         MOVE 16 TO CURR.
 AF15.
     MOVE 1 TO WORK-E.
     MOVE ZERO TO PAST-ZERO.
 AF20.
     IF INDENT = 5
         IF (FI (WORK-E) = "0" AND PAST-ZERO = ZERO)
         OR (FI (1) = " " AND FI (WORK-E) = " ")
             GO TO AF30.
     MOVE 1 TO PAST-ZERO.
     MOVE FI (WORK-E) TO ORR (CURR).
 AF25.
     IF WORK-E < WORK-F
         ADD 1 TO WORK-E
         ADD 1 TO CURR
         GO TO AF20.
     GO TO AF35.
 AF30.
     IF WORK-E < WORK-F
         ADD 1 TO WORK-E
         GO TO AF20.
 AF35.
```

```
     IF FI (WORK-F) = "."
         PERFORM WRITEIT
     ELSE
         ADD 2 TO CURR.
     MOVE SPACES TO FIELD.
 AF99.
     EXIT.
*
 WRITEIT SECTION.
 WR00.
     IF OUT-REC = SPACES
         GO TO WR10.
     MOVE 72 TO RECORD-LENGTH.
 WR05.
     IF ORR (RECORD-LENGTH) = SPACE
         SUBTRACT 1 FROM RECORD-LENGTH
         GO TO WR05.
     WRITE OUT-REC.
     MOVE SPACES TO OUT-REC.
 WR10.
     MOVE 7 TO CURR.
 WR99.
     EXIT.
*
 IO-SKEL SECTION.
 IO00.
     MOVE SPACES TO SKEL-REC.
     READ SKELETON AT END
         DISPLAY "END OF SKELETON FILE"
         GO TO IO99.
     IF NOT END-OF-DATA
         MOVE SKEL-REC TO OUT-REC
         PERFORM WRITEIT
         GO TO IO00.
 IO99.
     EXIT.
*
 START-DETAILS SECTION.
 SD00.
     PERFORM CLOSE-FM.
     OPEN INPUT FM.
     MOVE ZERO TO ST-FM-CLOSED.
     MOVE ZERO TO EOF-REPLY.
     PERFORM READ-DETAILS.
 SD99.
     EXIT.
*
 READ-DETAILS SECTION.
 RD00.
     READ FM NEXT AT END
         MOVE 1 TO EOF-REPLY
         PERFORM CLOSE-FM.
 RD99.
     EXIT.
*
 FIRST-DETAILS SECTION.
 FD00.
     PERFORM START-DETAILS.
     MOVE FM-FILE TO FILE-NAME.
```

```
    MOVE FM-TITLE TO FILE-TITLE.
    MOVE 1 TO PPAGE.
FD05.
    IF FM-KEY-FLAG = 1
        MOVE FM-FIELD TO KEY-NAME.
    IF FM-PAGE NUMERIC
        IF FM-PAGE > PPAGE
            MOVE FM-PAGE TO PPAGE.
    PERFORM READ-DETAILS.
    IF NOT END-OF-FILE
        GO TO FD05.
    PERFORM CLOSE-FM.
FD99.
    EXIT.
```

Produced FM Program

This is produced by the above program from a combination of the skeleton program and codings applied from the parameter file. The lines obtained from the skeleton are, of course, not noticeable as such in the actual program but have been prefixed by a vertical bar (|) in the following listing.

```
| IDENTIFICATION DIVISION.
  PROGRAM-ID. FMCREDIT.
|*
| ENVIRONMENT DIVISION.
| CONFIGURATION SECTION.
| SOURCE-COMPUTER. VAX-11.
| OBJECT-COMPUTER. VAX-11.
| SPECIAL-NAMES.
|      SWITCH 1 ON IS DELETE-OK
|      SWITCH 2 ON IS INSERT-OK
|      SWITCH 3 ON IS AMEND-OK.
|*
| INPUT-OUTPUT SECTION.
| FILE-CONTROL.
|*
|* the file MASTER is the indexed sequential
|*   file to be maintained. The file PRINTER
|*   is an audit trail of all actions on the file
|*
      SELECT CREDIT ASSIGN TO CREDITF
          ORGANIZATION IS INDEXED
          ACCESS IS DYNAMIC
          RECORD KEY IS CREDIT-JOB
          ALTERNATE RECORD KEY IS CREDIT-NOTE.
|     SELECT AUDIT ASSIGN TO PRINTER.
| DATA DIVISION.
| FILE SECTION.
  FD CREDIT.
  Ø1 CREDIT-RECORD.
      Ø3 CREDIT-JOB                PIC X(12).
      Ø3 CREDIT-DESC               PIC X(5Ø) OCCURS 3.
      Ø3 CREDIT-AMOUNT             PIC 9(7)V9(2).
      Ø3 CREDIT-NOTE               PIC 9(6).
      Ø3 CREDIT-FILLER             PIC X(1Ø).
| FD  AUDIT.
| Ø1  AUDIT-RECORD.
|     Ø3  AUDIT-NAME          PIC X(3Ø).
|     Ø3  AUDIT-VALUE-X       PIC X(5Ø).
|     Ø3  FILLER REDEFINES AUDIT-VALUE-X.
|         Ø5  AUDIT-VALUE-9  PIC -(17)9.
|         Ø5  FILLER        PIC X(32).
|     Ø3  FILLER REDEFINES AUDIT-VALUE-X.
|         Ø5  AUDIT-VALUE-R  PIC -(12)9.9(5).
|         Ø5  FILLER        PIC X(31).
```

```
    03  FILLER REDEFINES AUDIT-VALUE-X.
        05  FILLER          PIC X(20).
        05  AUDIT-HEAD.
          07  AUDIT-DAY       PIC 99.
          07  FILLER          PIC X.
          07  AUDIT-MONTH     PIC 99.
          07  FILLER          PIC X.
          07  AUDIT-YEAR      PIC 99.
          07  FILLER          PIC XXX.
          07  AUDIT-HOUR      PIC 99.
          07  FILLER          PIC X.
          07  AUDIT-MINUTE    PIC 99.
          07  FILLER          PIC X.
          07  AUDIT-SECOND    PIC 99.
          07  FILLER          PIC XXX.
          07  AUDIT-PAGE-LITERAL PIC XXXX.
          07  AUDIT-PAGE      PIC ZZ9.
WORKING-STORAGE SECTION.
01  THIS-KEY              PIC X(20) VALUE "job".
01  DECIMALS               PIC 9.
01  ROW                    PIC 99.
01  COLUMN                PIC 99.
01  ST-AUDIT-RECORD        PIC X(80).
01  ST-TITLE               PIC X(40) VALUE SPACES.
01  LINE-COUNT             PIC 99 COMP VALUE 100.
01  PAGE-COUNT             PIC 999 COMP VALUE 1.
01  ST-DATE-TIME.
    03  ST-DATE            PIC 9(6).
    03  FILLER REDEFINES ST-DATE.
        05  ST-YEAR        PIC 99.
        05  ST-MONTH       PIC 99.
        05  ST-DAY         PIC 99.
    03  ST-TIME            PIC 9(8).
    03  FILLER REDEFINES ST-TIME.
        05  ST-HOUR        PIC 99.
        05  ST-MINUTE      PIC 99.
        05  ST-SECOND      PIC 99.
        05  FILLER         PIC XX.
01  AUDIT-ADVANCE                PIC 9999 COMP.
01  FUNCTION               PIC X(6).
    88  INSERT-RECORD VALUE "Insert".
    88  SHOW-RECORD VALUE "Show  ".
    88  AMEND-RECORD VALUE "Amend ".
    88  DELETE-RECORD VALUE "Delete".
    88  NO-FUNCTION VALUE SPACES.
01  REPLY                  PIC X.
    88  NO-KEY VALUE "1".
    88  NO-RECORD VALUE "2".
    88  YES-RECORD VALUE "3".
    88  APPLY-CHANGE VALUE "4".
01  KEY-STATUS             PIC X VALUE SPACE.
    88  NEW-KEY VALUE SPACE.
01  SCREEN-PAGE            PIC 9 COMP.
01  ST-SIZE                PIC 99 COMP.
01  INPUT-S.
    88  NO-INPUT VALUE SPACES.
    03  IXX                PIC X OCCURS 60.
01  OUTPUT-I               PIC -(17)9.
01  OUTPUT-R               PIC -(12)9.9(5).
```

```
 01  LOWER-I              PIC 9(18)-.
 01  UPPER-I              PIC 9(18)-.
 01  LOWER-R          PIC S9(13)V9(5) COMP.
 01  UPPER-R          PIC S9(13)V9(5) COMP.
 01  W-JULIAN-DATE-WORK-AREA.
     03  W-JD-DATE                PIC 9(6).
     03  FILLER REDEFINES W-JD-DATE.
         05  W-J-DAY              PIC 99.
         05  W-J-MONTH            PIC 99.
         05  W-J-YEAR             PIC 99.
     03  W-J-JULIAN               PIC 9(6) COMP.
 01  INPUT-D          PIC 9(6) COMP.
 01  LOWER-D          PIC 9(6) COMP.
 01  UPPER-D          PIC 9(6) COMP.
 01  SUBSCRIPT        PIC 999 COMP.
 01  WORK-1           PIC 99 COMP.
 01  WORK-2           PIC 99 COMP.
 01  ERROR-MESSAGE        PIC X(78).
 01  INPUT-I          PIC S9(18) COMP.
 01  INPUT-R          PIC S9(13)V9(5) COMP.
*
 PROCEDURE DIVISION.
 CONTROLX SECTION.
 C00.
     PERFORM INITIALX.
     PERFORM MAINFLOW.
     PERFORM CLOSEDOWN.
 C99.
     STOP RUN.
*
 INITIALX SECTION.
*
* open the files; clear the screen;
*   display the program title on
*   the vdu; print the audit headings
*
 I00.
     OPEN I-O CREDIT.
     OPEN EXTEND AUDIT.
     MOVE 1 TO ROW.
     CALL "CLEARVDU" USING ROW.
     MOVE "Credit Notes"
         TO ST-TITLE.
     DISPLAY ST-TITLE WITH NO ADVANCING.
 IN99.
     EXIT.
*
* get the function option
*   I = insert a new record
*   A = amend an existing record
*   S = look at (show) a record
*   D = delete an existing record
*
* then perform the relevant screen control
*
 MAINFLOW SECTION.
 M00.
     PERFORM GET-FUNCTION.
     IF NO-FUNCTION
```

```
            GO TO M99.
 M05.
        PERFORM GET-KEY.
        IF NO-KEY
            GO TO M00.
        MOVE 0 TO SCREEN-PAGE.
        IF INSERT-RECORD
            PERFORM INSERT-R
        ELSE
        IF AMEND-RECORD
            PERFORM AMEND-R
        ELSE
        IF SHOW-RECORD
            PERFORM SHOW-R
        ELSE
            PERFORM DELETE-R.
        IF THIS-KEY NOT ="job"
            MOVE 23 TO ROW
            MOVE 1 TO COLUMN
            CALL "CURSORTO" USING ROW, COLUMN
            DISPLAY "Press return to continue" WITH NO ADVANCING
            MOVE 1 TO ST-SIZE
            CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE
            MOVE 4 TO ROW
            CALL "CLEARVDU" USING ROW.
        GO TO M05.
 M99.
        EXIT.
*
* end the audit report and close files before the end of the
* run
*
 CLOSEDOWN SECTION.
 C00.
        MOVE "End of Audit Report" TO AUDIT-RECORD.
        MOVE 2 TO AUDIT-ADVANCE.
        PERFORM AUDIT-PRINT.
        CLOSE CREDIT.
        CLOSE AUDIT.
 C99.
        EXIT.
*
* common routine for writing a record to the audit file
*
 AUDIT-PRINT SECTION.
 AU00.
        IF LINE-COUNT > 55
            PERFORM AUDIT-HEADINGS.
        WRITE AUDIT-RECORD AFTER ADVANCING AUDIT-ADVANCE LINES.
        MOVE SPACES TO AUDIT-RECORD.
        ADD AUDIT-ADVANCE TO LINE-COUNT.
        MOVE 1 TO AUDIT-ADVANCE.
 AU99.
        EXIT.
*
* accept the key from the screen then check the
*    file that the record is absent (for insert)
*    or present (for amend, show and delete).
*
```

```
GET-KEY SECTION.
GE00.
    IF THIS-KEY = "job"
        PERFORM DISPLAY-STATICS-JOB
    ELSE
    IF THIS-KEY = "note number"
        PERFORM DISPLAY-STATICS-NOTE
    ELSE
        GO TO GE90.
GE05.
    MOVE ZERO TO REPLY.
    IF THIS-KEY = "job"
        PERFORM INPUT-JOB
    ELSE
    IF THIS-KEY = "note number"
        PERFORM INPUT-NOTE
    ELSE
        GO TO GE90.
    IF NO-INPUT
        IF INSERT-RECORD
            MOVE 1 TO REPLY
            GO TO GE99
        ELSE
            GO TO GE40.
    IF INPUT-S NOT = "+"
        PERFORM READ-VIA-INDEX
        GO TO GE20.
    IF INSERT-RECORD
        GO TO GE25.
    IF NEW-KEY
        PERFORM READ-VIA-INDEX.
    READ CREDIT NEXT AT END
        GO TO GE25.
    MOVE 3 TO REPLY.
GE20.
    PERFORM DISPLAY-JOB.
    IF INSERT-RECORD
        GO TO GE99.
    IF YES-RECORD
        GO TO GE99.
GE25.
    MOVE "No such record" TO ERROR-MESSAGE.
    CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
        INPUT-S.
    GO TO GE05.
GE40.
    MOVE 23 TO ROW.
    MOVE 1 TO COLUMN.
    CALL "CURSORTO" USING ROW, COLUMN.
    DISPLAY "Which key" WITH NO ADVANCING.
    MOVE 20 TO ST-SIZE.
    MOVE 12 TO COLUMN.
    MOVE ZERO TO REPLY.
    CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE.
    IF NO-INPUT
        MOVE 1 TO REPLY
        GO TO GE99.
    IF INPUT-S = "job" OR "note number"
        NEXT SENTENCE
```

```
        ELSE
            GO TO GE45.
        IF INPUT-S NOT = THIS-KEY
            MOVE INPUT-S TO THIS-KEY
            MOVE SPACE TO KEY-STATUS.
        MOVE 4 TO ROW.
        CALL "CLEARVDU" USING ROW.
        GO TO GE00.
    GE45.
        MOVE "One of: job, note number" TO ERROR-MESSAGE.
        CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
           INPUT-S.
        GO TO GE40.
    GE90.
        MOVE "Invalid key selection contents" TO ERROR-MESSAGE.
        CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
           INPUT-S.
        GO TO FAIL-END.
    GE99.
*
* this reads a file via one if its indices
*
    READ-VIA-INDEX SECTION.
    RV00.
        MOVE "X" TO KEY-STATUS.
        MOVE 3 TO REPLY.
        IF THIS-KEY = "job"
            GO TO RV-JOB.
        IF THIS-KEY = "note number"
            GO TO RV-NOTE.
        GO TO RV90.
    RV-JOB.
        READ CREDIT KEY IS CREDIT-JOB INVALID KEY
            MOVE 2 TO REPLY.
        GO TO RV99.
    RV-NOTE.
        READ CREDIT KEY IS CREDIT-NOTE INVALID KEY
            MOVE 2 TO REPLY.
        GO TO RV99.
    RV90.
        MOVE "Invalid key selection contents" TO ERROR-MESSAGE.
        CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
           INPUT-S.
        GO TO FAIL-END.
    RV99.
        EXIT.
*
* this controls inputting a new record onto the file. It
* allows initial input of fields, amending of any errors,
* the option to abandon the insertion, writing of the
* record to the file and auditing the new record's contents.
*
    INSERT-R SECTION.
    IN00.
        MOVE "Insert a new record" TO AUDIT-RECORD.
        MOVE 3 TO AUDIT-ADVANCE.
        PERFORM AUDIT-PRINT.
        PERFORM INPUT-FIELDS.
        PERFORM AMEND-FIELDS.
```

```
      PERFORM INQUIRE-OK.
      IF NOT APPLY-CHANGE
          GO TO IN90.
      PERFORM AUDIT-CREDIT.
      WRITE CREDIT-RECORD INVALID KEY
          MOVE "WRITE FAIL" TO AUDIT-RECORD
          PERFORM AUDIT-PRINT
          GO TO FAIL-END.
      MOVE "Details inserted" TO ERROR-MESSAGE.
      GO TO IN95.
 IN90.
      PERFORM AUDIT-ABANDONED.
      MOVE "Details not inserted" TO ERROR-MESSAGE.
 IN95.
      CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
          INPUT-S.
 IN99.
*
*  this controls the amending of an existing record. It
*   displays the first page,  allows amending of
*   specific field values on this or any subsequent
*   pages, gives the option to abandon the amendments,
*   writes the new record values and audits the new
*   record values.
*
 AMEND-R SECTION.
 AM00.
      MOVE "Amend an existing record" TO AUDIT-RECORD.
      MOVE 3 TO AUDIT-ADVANCE.
      PERFORM AUDIT-PRINT.
      PERFORM AUDIT-CREDIT.
      MOVE 1 TO SCREEN-PAGE.
      PERFORM SHOW-FIELDS.
      PERFORM AMEND-FIELDS.
      PERFORM INQUIRE-OK.
      IF NOT APPLY-CHANGE
          GO TO AM90.
      PERFORM AUDIT-CREDIT.
      REWRITE CREDIT-RECORD INVALID KEY
          MOVE "REWRITE FAIL" TO AUDIT-RECORD
          PERFORM AUDIT-PRINT
          GO TO FAIL-END.
      MOVE "Details updated" TO ERROR-MESSAGE.
      GO TO AM95.
 AM90.
      PERFORM AUDIT-ABANDONED.
      MOVE "Details retained" TO ERROR-MESSAGE.
 AM95.
      CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
          INPUT-S.
 AM99.
      EXIT.
*
* this section steps through, one at a time, the various pages
*  of the record's values and audits the record's contents
*
 SHOW-R SECTION.
 SH00.
      MOVE "Show an existing record" TO AUDIT-RECORD.
```

```
|       MOVE 3 TO AUDIT-ADVANCE.
|       PERFORM AUDIT-PRINT.
|       PERFORM SHOW-FIELDS.
        PERFORM AUDIT-CREDIT.
| SH99.
|       EXIT.
|*
|*  this section shows each page of information on the
|*   record to be deleted, allows the abandonment of the
|*   deletion, deletes the record and audits the late
|*   record's contents
|*
| DELETE-R SECTION.
| DE00.
|       MOVE "Delete an existing record" TO AUDIT-RECORD.
|       MOVE 3 TO AUDIT-ADVANCE.
|       PERFORM AUDIT-PRINT.
|       PERFORM SHOW-FIELDS.
        PERFORM AUDIT-CREDIT.
|       PERFORM INQUIRE-OK.
|       IF NOT APPLY-CHANGE
|           GO TO DE90.
        DELETE CREDIT INVALID KEY
|           MOVE "DELETE FAIL" TO AUDIT-RECORD
|           PERFORM AUDIT-PRINT
|           GO TO FAIL-END.
|       MOVE "Details deleted" TO ERROR-MESSAGE.
|       GO TO DE95.
| DE90.
|       PERFORM AUDIT-ABANDONED.
|       MOVE "Details retained" TO ERROR-MESSAGE.
| DE95.
|       CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
|           INPUT-S.
| DE99.
|       EXIT.
|*
|*  this asks on the 23rd line whether the change is to be
|*    applied to the file. It is asked (i) after amendment of
|*    the fields in the insert function (ii) after amendment
|*    of the fields in the amend function (iii) after viewing
|*    all the contents of the record in the delete function.
|*  It sets the variable REPLY.
|*
| INQUIRE-OK SECTION.
| IQ00.
|       MOVE 23 TO ROW.
|       MOVE 1 TO COLUMN.
|       CALL "CURSORTO" USING ROW, COLUMN.
|       DISPLAY "Apply the change (Y=Yes)" WITH NO ADVANCING.
|       MOVE 28 TO COLUMN.
|       CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE.
|       CALL "CLEARVDU" USING ROW.
|       IF IXX (1) = "Y" OR IXX (1) = "y"
|           MOVE 4 TO REPLY
|       ELSE
|           MOVE ZERO TO REPLY.
| IQ99.
|       EXIT.
```

```
|*
|* this inputs and verifies the function selected and prints
|*  the function selected on the screen
|*
| GET-FUNCTION SECTION.
| GFØØ.
|     MOVE 23 TO ROW.
| GFØ5.
|     MOVE 1 TO COLUMN.
|     CALL "CURSORTO" USING ROW, COLUMN.
|     IF DELETE-OK
|         DISPLAY "I,A,S or D - which one" WITH NO ADVANCING
|     ELSE
|     IF INSERT-OK
|         DISPLAY "I,A or S - which one" WITH NO ADVANCING
|     ELSE
|     IF AMEND-OK
|         DISPLAY "A or S - which one" WITH NO ADVANCING
|     ELSE
|         DISPLAY "S or return - which one" WITH NO ADVANCING.
|     MOVE 25 TO COLUMN.
|     MOVE 1 TO ST-SIZE.
|     CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE.
|     IF IXX (1) = " "
|         MOVE SPACES TO FUNCTION
|         GO TO GF99.
|     IF (IXX (1) = "I" OR "i") AND (DELETE-OK OR INSERT-OK)
|         MOVE "Insert" TO FUNCTION
|         MOVE "job" TO THIS-KEY
|         GO TO GF1Ø.
|     IF (IXX (1) = "A" OR "a") AND (DELETE-OK OR INSERT-OK OR
|     AMEND-OK)
|         MOVE "Amend " TO FUNCTION
|         GO TO GF1Ø.
|     IF IXX (1) = "S" OR "s"
|         MOVE "Show  " TO FUNCTION
|         GO TO GF1Ø.
|     IF (IXX (1) = "D" OR "d") AND DELETE-OK
|         MOVE "Delete" TO FUNCTION
|         GO TO GF1Ø.
|     IF DELETE-OK
|         MOVE "I=insert, A=amend, S=Show, D=delete"
|         TO ERROR-MESSAGE
|     ELSE
|     IF INSERT-OK
|         MOVE "I=insert, A=amend, S=Show"
|         TO ERROR-MESSAGE
|     ELSE
|     IF AMEND-OK
|         MOVE "A=amend, S=Show"
|         TO ERROR-MESSAGE
|     ELSE
|         MOVE "S=Show"
|         TO ERROR-MESSAGE.
|     CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
|         INPUT-S.
|     GO TO GFØ5.
| GF1Ø.
|     MOVE 23 TO ROW.
```

```
     CALL "CLEARVDU" USING ROW.
     MOVE 2 TO ROW.
     MOVE 70 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY FUNCTION WITH NO ADVANCING.
 GF99.
     EXIT.
*
* This prints a message on the audit trail to signal the
*  last option (insert, amend or delete) was abandoned
*  with no change to the file.
*
 AUDIT-ABANDONED SECTION.
 AUDI00.
     MOVE "** function abandoned **" TO AUDIT-RECORD.
     MOVE 2 TO AUDIT-ADVANCE.
     PERFORM AUDIT-PRINT.
 AUDI99.
     EXIT.
*
* head the audit print and reset the line and page
*   counters
 AUDIT-HEADINGS SECTION.
 AH00.
     MOVE AUDIT-RECORD TO ST-AUDIT-RECORD.
     MOVE SPACES TO AUDIT-RECORD.
     MOVE ST-TITLE TO AUDIT-RECORD.
     MOVE "  /  /      :  :      Page" TO AUDIT-HEAD.
     MOVE PAGE-COUNT TO AUDIT-PAGE.
     ACCEPT ST-DATE FROM DATE.
     ACCEPT ST-TIME FROM TIME.
     MOVE ST-YEAR TO AUDIT-YEAR.
     MOVE ST-MONTH TO AUDIT-MONTH.
     MOVE ST-DAY TO AUDIT-DAY.
     MOVE ST-HOUR TO AUDIT-HOUR.
     MOVE ST-MINUTE TO AUDIT-MINUTE.
     MOVE ST-SECOND TO AUDIT-SECOND.
     WRITE AUDIT-RECORD AFTER ADVANCING PAGE.
     MOVE SPACES TO AUDIT-RECORD.
     WRITE AUDIT-RECORD AFTER ADVANCING 2 LINES.
     MOVE 1 TO LINE-COUNT.
     ADD 1 TO PAGE-COUNT.
     MOVE ST-AUDIT-RECORD TO AUDIT-RECORD.
*
* this forces the program to fail with an error being
*  returned to the operating system.
*
 FAIL-END SECTION.
 FA00.
     MOVE ERROR-MESSAGE TO AUDIT-RECORD.
     PERFORM AUDIT-PRINT.
     MOVE "PROGRAM RUN ABANDONED" TO AUDIT-RECORD, ERROR-MESSAGE.
     CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
         INPUT-S.
     PERFORM AUDIT-PRINT.
     MOVE ZERO TO WORK-1.
     DIVIDE 2 BY WORK-1 GIVING WORK-2.
     PERFORM CLOSEDOWN.
     STOP "PROGRAM RUN ABANDONED".
```

```
| FA99.
|     EXIT.
|*
|* this function controls global inputting of
|*   all fields
| INPUT-FIELDS SECTION.
| INPØØ.
      MOVE 1 TO SCREEN-PAGE.
      PERFORM NEW-PAGE.
      PERFORM INPUT-DESC.
      PERFORM INPUT-AMOUNT.
      PERFORM INPUT-NOTE.
      MOVE SPACES TO CREDIT-FILLER.
| INP99.
|     EXIT.
|*
|* this function controls selection of amendment
|*   of an individual field
|*
| AMEND-FIELDS SECTION.
| AMEØØ.
|     MOVE 23 TO ROW.
|     MOVE 1 TO COLUMN.
|     CALL "CURSORTO" USING ROW, COLUMN.
|     DISPLAY "Which Field" WITH NO ADVANCING.
|     MOVE 14 TO COLUMN.
      MOVE ZERO TO LOWER-I.
      MOVE 3 TO UPPER-I.
|     CALL "INPUTI" USING ROW, COLUMN, LOWER-I, UPPER-I,
|         INPUT-I.
|     MOVE 23 TO ROW.
|     CALL "CLEARVDU" USING ROW.
|     IF INPUT-I = ZERO
|         GO TO AME99.
      IF INPUT-I NOT > 3
          MOVE 1 TO WORK-1.
      IF WORK-1 NOT = SCREEN-PAGE
          MOVE WORK-1 TO SCREEN-PAGE
          PERFORM SHOW-FIELDS
          GO TO AMEØØ.
      IF INPUT-I = 1
          PERFORM INPUT-DESC
      ELSE
      IF INPUT-I = 2
          PERFORM INPUT-AMOUNT
      ELSE
      IF INPUT-I = 3
          PERFORM INPUT-NOTE.
      GO TO AMEØØ.
| AME99.
|     EXIT.
|*
|* this function allows the display of either
|*   the first page only (for amend and delete)
|*   or all screens (for show).
|*
| SHOW-FIELDS SECTION.
| SHOØØ.
|     IF SHOW-RECORD OR DELETE-RECORD
```

```
|         MOVE 1 TO SCREEN-PAGE.
| SHOØ5.
|     PERFORM NEW-PAGE.
      IF SCREEN-PAGE = 1
          PERFORM DISPLAY-DESC
          PERFORM DISPLAY-AMOUNT
          PERFORM DISPLAY-NOTE .
|     IF NOT (SHOW-RECORD
|     OR DELETE-RECORD)
      OR SCREEN-PAGE = 1
|         GO TO SHO99.
|     MOVE 23 TO ROW.
|     MOVE 1 TO COLUMN.
|     CALL "CURSORTO" USING ROW, COLUMN.
|     DISPLAY "Next Page" WITH NO ADVANCING.
|     ACCEPT INPUT-S.
|     CALL "CLEARVDU" USING ROW.
|     ADD 1 TO SCREEN-PAGE.
|     GO TO SHOØ5.
| SHO99.
|     EXIT.
|*
|* the following sections control the displaying of
|*    background screen information relating to
|*    the master file fields. The following are
|*    displayed: (i) the field amending number
|*    (if the field is amendable) (ii) a colon
|*    between the field number and the name
|*    (iii) the field display name (iii) the
|*    colon between the name and input/display
|*    area and a colon immediately below this
|*    for every line the input/display area involves
|*
|*
|*
|*
 DISPLAY-STATICS-JOB SECTION.
 DS-JOB-ØØ.
     MOVE 2 TO ROW.
     MOVE 3 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY " job" WITH NO ADVANCING.
     MOVE 29 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY ":" WITH NO ADVANCING.
     MOVE 42 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY ":" WITH NO ADVANCING.
 DS-JOB-99.
     EXIT.
*
 DISPLAY-STATICS-DESC SECTION.
 DS-DESC-ØØ.
     MOVE 4 TO ROW.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY 1 WITH NO ADVANCING.
     MOVE 3 TO COLUMN.
```

```
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY " description" WITH NO ADVANCING.
           MOVE 1 TO SUBSCRIPT.
       DS-DESC-Ø5.
           MOVE 29 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY ":" WITH NO ADVANCING.
           MOVE 8Ø TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY ":" WITH NO ADVANCING.
           IF SUBSCRIPT <3
               ADD 1 TO ROW
               ADD 1 TO SUBSCRIPT
               GO TO DS-DESC-Ø5.
       DS-DESC-99.
           EXIT.
      *
       DISPLAY-STATICS-AMOUNT SECTION.
       DS-AMOUNT-ØØ.
           MOVE 7 TO ROW.
           MOVE 1 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY 2 WITH NO ADVANCING.
           MOVE 3 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY " amount" WITH NO ADVANCING.
           MOVE 29 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY ":" WITH NO ADVANCING.
           MOVE 4Ø TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY ":" WITH NO ADVANCING.
       DS-AMOUNT-99.
           EXIT.
      *
       DISPLAY-STATICS-NOTE SECTION.
       DS-NOTE-ØØ.
           MOVE 8 TO ROW.
           MOVE 1 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY 3 WITH NO ADVANCING.
           MOVE 3 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY " note number" WITH NO ADVANCING.
           MOVE 29 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY ":" WITH NO ADVANCING.
           MOVE 36 TO COLUMN.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY ":" WITH NO ADVANCING.
       DS-NOTE-99.
           EXIT.
      |*
      |* the following sections control the inputting
      |*   and verification of the individual fields
      |*   on the master file.
      |*
      |*
      |*
```

```
|*
|*
 INPUT-JOB SECTION.
 I-JOB-ØØ.
     MOVE 2 TO ROW.
     MOVE 3Ø TO COLUMN.
 I-JOB-Ø5.
     MOVE 12 TO ST-SIZE.
     CALL "INPUTS" USING ROW, COLUMN,
     INPUT-S, ST-SIZE.
     IF THIS-KEY = "job"
     AND (INPUT-S = "+" OR SPACES)
         GO TO I-JOB-99.
     MOVE INPUT-S TO CREDIT-JOB .
 I-JOB-99.
     EXIT.
*
 INPUT-DESC SECTION.
 I-DESC-ØØ.
     MOVE 4 TO ROW.
     MOVE 3Ø TO COLUMN.
     MOVE 1 TO SUBSCRIPT.
 I-DESC-Ø5.
     MOVE 5Ø TO ST-SIZE.
     CALL "INPUTS" USING ROW, COLUMN,
     INPUT-S, ST-SIZE.
     MOVE INPUT-S TO CREDIT-DESC (SUBSCRIPT) .
 I-DESC-1Ø.
     IF SUBSCRIPT < 3
         ADD 1 TO SUBSCRIPT
         ADD 1 TO ROW
         GO TO I-DESC-Ø5.
 I-DESC-99.
     EXIT.
*
 INPUT-AMOUNT SECTION.
 I-AMOUNT-ØØ.
     MOVE 7 TO ROW.
     MOVE 3Ø TO COLUMN.
     MOVE .Ø1 TO LOWER-R .
     MOVE 9999999.99 TO UPPER-R .
     MOVE 2 TO DECIMALS.
 I-AMOUNT-Ø5.
     CALL "INPUTR" USING ROW, COLUMN,
     LOWER-R, UPPER-R, INPUT-R,
     DECIMALS.
     MOVE INPUT-R TO CREDIT-AMOUNT .
 I-AMOUNT-99.
     EXIT.
*
 INPUT-NOTE SECTION.
 I-NOTE-ØØ.
     MOVE 8 TO ROW.
     MOVE 3Ø TO COLUMN.
     MOVE 1 TO LOWER-I .
     MOVE 999999 TO UPPER-I .
 I-NOTE-Ø5.
     CALL "INPUTIK" USING ROW, COLUMN,
     LOWER-I, UPPER-I, INPUT-I, INPUT-S.
```

```
        IF THIS-KEY = "note number"
        AND (INPUT-S = "+" OR SPACES)
            GO TO I-NOTE-99.
        MOVE INPUT-I TO CREDIT-NOTE .
    I-NOTE-99.
        EXIT.
*
* the following sections allow the display of the
*   contents of individual fields on the master
*   file.
*
*
*
*
*
    DISPLAY-JOB SECTION.
    D-JOB-00.
        MOVE 2 TO ROW.
        MOVE 30 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY CREDIT-JOB WITH NO ADVANCING.
    D-JOB-99.
        EXIT.
*
    DISPLAY-DESC SECTION.
    D-DESC-00.
        MOVE 4 TO ROW.
        MOVE 30 TO COLUMN.
        MOVE 1 TO SUBSCRIPT.
    D-DESC-05.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY CREDIT-DESC (SUBSCRIPT) WITH NO ADVANCING.
        IF SUBSCRIPT < 3
            ADD 1 TO SUBSCRIPT
            ADD 1 TO ROW
            GO TO D-DESC-05.
    D-DESC-99.
        EXIT.
*
    DISPLAY-AMOUNT SECTION.
    D-AMOUNT-00.
        MOVE 7 TO ROW.
        MOVE 30 TO COLUMN.
        IF CREDIT-AMOUNT NUMERIC
            MOVE CREDIT-AMOUNT TO OUTPUT-R
            CALL "OUTPUTR" USING ROW, COLUMN, OUTPUT-R
        ELSE
            CALL "CURSORTO" USING ROW, COLUMN
            DISPLAY CREDIT-AMOUNT WITH NO ADVANCING.
    D-AMOUNT-99.
        EXIT.
*
    DISPLAY-NOTE SECTION.
    D-NOTE-00.
        MOVE 8 TO ROW.
        MOVE 30 TO COLUMN.
        IF CREDIT-NOTE NUMERIC
            MOVE CREDIT-NOTE TO OUTPUT-I
            CALL "OUTPUTI" USING ROW, COLUMN, OUTPUT-I
```

```
            ELSE
                CALL "CURSORTO" USING ROW, COLUMN
                DISPLAY CREDIT-NOTE WITH NO ADVANCING.
        D-NOTE-99.
            EXIT.
      *
      * this controls the displaying of the background
      *   details on a selected screen page.
      *
       NEW-PAGE SECTION.
       NP00.
            MOVE 4 TO ROW.
            CALL "CLEARVDU" USING ROW.
            IF SCREEN-PAGE = 1
                PERFORM DISPLAY-STATICS-DESC
                PERFORM DISPLAY-STATICS-AMOUNT
                PERFORM DISPLAY-STATICS-NOTE .
       NP99.
            EXIT.
      *
      * this prints the contents of the MASTER file
      *   fields and is used by the functions selected
      *   to show the record as it stands (show, amend,
      *   delete), show the changed record (amend),
      *   show the new record  (input).
      *
        AUDIT-CREDIT SECTION.
       AUD00.
            MOVE 2 TO AUDIT-ADVANCE.
            MOVE "job" TO AUDIT-NAME.
            MOVE CREDIT-JOB TO AUDIT-VALUE-X.
            PERFORM AUDIT-PRINT.
            MOVE "description" TO AUDIT-NAME.
            MOVE 1 TO SUBSCRIPT.
        AUDIT-DESC.
            MOVE CREDIT-DESC (SUBSCRIPT) TO AUDIT-VALUE-X.
            PERFORM AUDIT-PRINT.
            IF SUBSCRIPT NOT = 3
                ADD 1 TO SUBSCRIPT
                GO TO AUDIT-DESC .
            MOVE "amount" TO AUDIT-NAME.
            IF CREDIT-AMOUNT NUMERIC
                MOVE CREDIT-AMOUNT TO AUDIT-VALUE-R
            ELSE
                MOVE "contents undefined" TO AUDIT-VALUE-X.
            PERFORM AUDIT-PRINT.
            MOVE "note number" TO AUDIT-NAME.
            IF CREDIT-NOTE NUMERIC
                MOVE CREDIT-NOTE TO AUDIT-VALUE-9
            ELSE
                MOVE CREDIT-NOTE TO AUDIT-VALUE-X.
            PERFORM AUDIT-PRINT.
            MOVE "FILLER" TO AUDIT-NAME.
            MOVE CREDIT-FILLER TO AUDIT-VALUE-X.
            PERFORM AUDIT-PRINT.
       AUD99.
            EXIT.
```

Produced Documentation

This is not included in the s-algol generator (above) but could be if required. It produces audit print layouts, vdu layouts and operating instructions.

For an example of unformatted documentation see the tpg suite in the next section of this chapter.

Program Documentation

program: FMCREDIT.COB Audit Print Layout

```
Credit Notes File Maintenance          DD/MM/YY  HH/MM/SS   Page Z9

Insert a new record

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

Amend an existing record

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

Show an existing record

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

Delete an existing record

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

Insert a new record
```

```
** function abandoned **


Amend an existing record

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

** function abandoned **


Delete an existing record

job                 XXXXXXXXXXX
description         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
amount                    9999999.99999
note number                     999999
FILLER              XXXXXXXXXX

** function abandoned **


End of Audit Report
```

Program Documentation

program: FMCREDIT.COB Screen Layouts

```
Credit Notes File Maintenance
   job                    :XXXXXXXXXXX:                  XXXXXX

1  description    :XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:
                  :XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:
                  :XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:
2  amount         :9999999.99:
3  note number    :999999:



<line 23 - prompts>
<line 24 - errors>
```

File Maintenance Operating Instructions
Credit Notes

This program allows you to add new details, view (show) present details, amend present details and remove old details from the Credit Notes file. The program automatically produces a print of all actions which is accumulated and printed later as an audit of all activity on the file.

Records of information are identified by the job (primary key) , note number . Amendments, deletions and viewing can take place via any key, but insertions only via the primary key.

Function

From this prompt at the bottom of the screen you select whether to insert new records (I), amend existing records (A), view (S) existing records or delete records (D). If you do not enter one of these the program will (i) signal an error and give an explanation of the entry required, returning you to enter the function again (ii) if you supply no entry the program run will end.

The function selected is displayed on the top right corner of the screen as insert, amend, show or delete appropriately.

On starting the program the key being used is the job .

job

At this prompt you enter the record key. One of the following actions can occur here (i) if the key is not on the file already and the option to insert has been selected, or if the key is on the file and amend, show or delete has been selected the program will continue as indicated below (ii) if the record does not exist and the option to amend, show or delete is selected then an error is displayed and you must re-enter the key (iii) if no entry is given the program asks you to select 'Which key'. The options are job (primary key) , note number . If you make no selection here then the current key is still used and you return to the 'function' prompt above.

Insert a new record

The field names are displayed. You enter the correct values (see 'Fields and their Values' below). If any of the fields are incorrect then an explanatory error is displayed and you must re-enter the field's value.

Once all the fields are in, the program will prompt you on the bottom line with Which field. Enter the number (left of field name on screen) of the field with an incorrect value. Fields with no number cannot be amended.

Once all the details are correct you press return at the Which Field prompt. The program replaces this with Apply the Change (Y=Yes). Enter 'Y' (upper or lower case) and the new record is added, otherwise the details are rejected.

The program returns you to the key entry above for a new record.

Amend an existing record

The screen is displayed complete with all details. The program prompts with Which Field on the bottom line. To amend a field enter the number (far left of field). Fields with no number cannot be amended.

Once all the details have been corrected as required you can either apply the changes to the file or abandon the amendment. Enter return to the Which Field prompt and the program asks Apply the Change (Y=Yes) Enter 'Y' (upper or lower case) to change the file. Any other entry abandons the changes.

If the key being used not is the job the program prompts "Press return to continue". Press Return.

The program returns you to the key entry above for another record.

Show an existing record

The screen is displayed. Once all the detail lines have been viewed the program returns to key entry to allow viewing of another record.

If the key being used is not the job the program prompts "Press return to continue". Press Return.

Delete an existing record

The screen is displayed. Once all the details have been viewed you can opt to delete the record or not. The program prompts Apply the Change (Y=Yes) on the bottom line. Enter 'Y' (in upper or lower case) to delete the record, otherwise the deletion is abandoned.

If the key being used is not the job the program prompts "Press return to continue". Press Return.

The program returns you to enter another key.

Fields and their Values

job (this is the primary key) : up to 12 characters , on 1 line of the screen.

description : up to 50 characters , on 3 lines of the screen.

amount : up to 7 numerics before the decimal point 2 numerics after the decimal point , ranging from .01 to 9999999.99 , on 1 line of the screen.

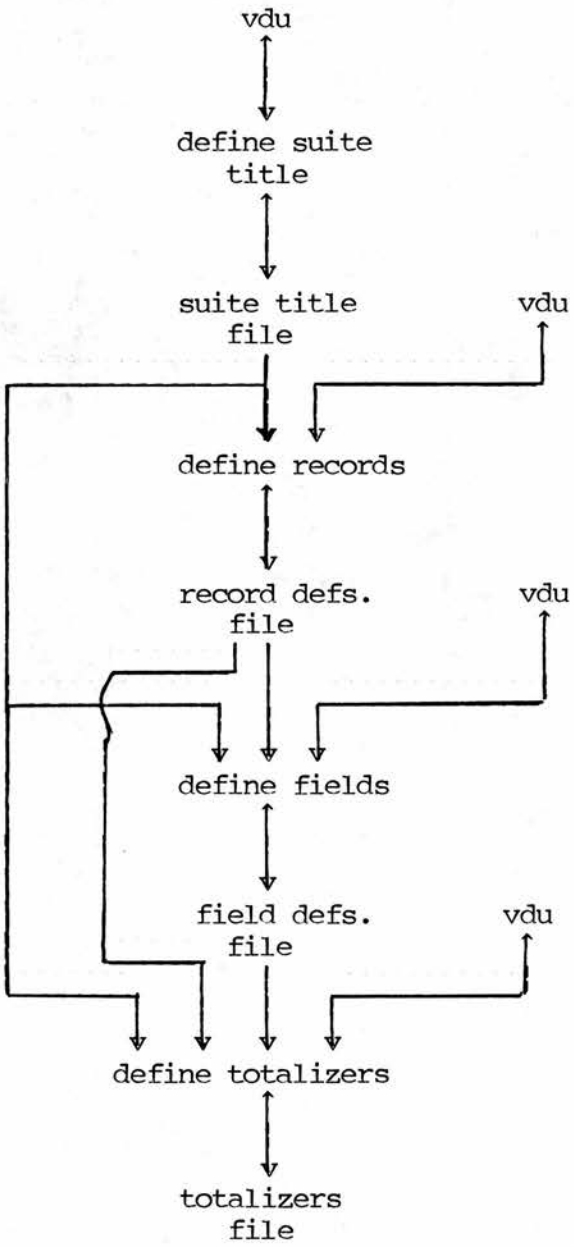
note number (this is a key) : up to 6 numerics , ranging from 1 to 999999 , on 1 line of the screen.

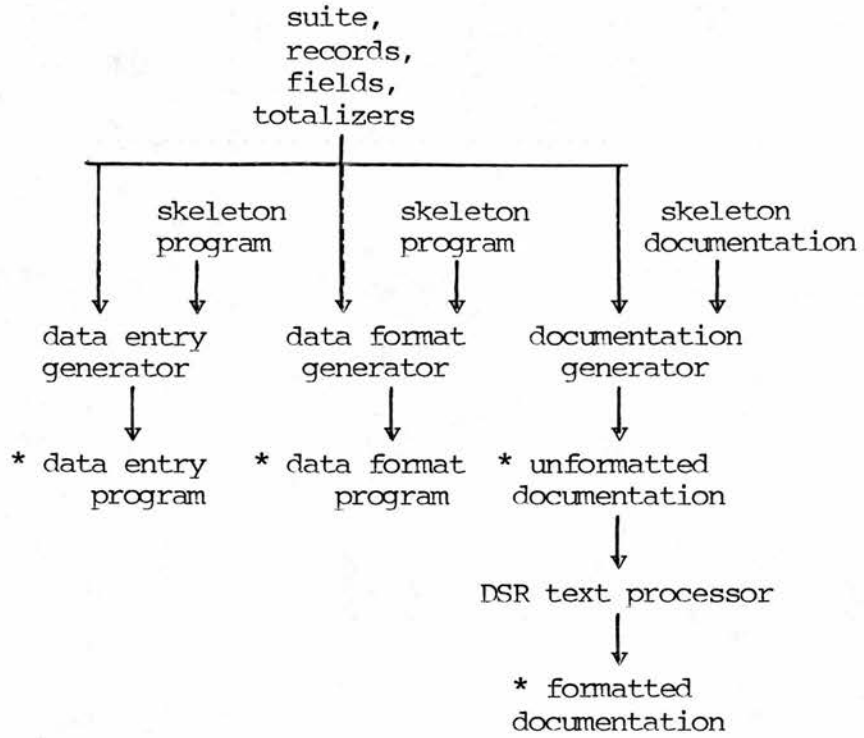
4. The COBOL Transaction Processing Generator

This represents only the output from the generator. Full details of the VAX DCL command procedure to run the generator, t.p. suite definition program, record definition program, field definition program, totalizers definition program, t.p. input program producer, t.p. output program producer, documentation producer, documentation and associated files are held in an unbound appendix and tape at the department.

Although this generator is much more complex than the fm generator, the logic is basically the same. I have not included it to relieve the reader having to plough his way through 300+ pages of COBOL code.

figure 4.17 - COBOL tpg suite overview





* indicates these are included as examples below.

The generator:

(i) Asks for the suite name. It then checks whether a suite already exists with this name and notifies the operator.

(ii) then asks the suite parameters.

(ii) asks the record parameters.

(iii) asks the fields' parameters.

(iv) asks any total checking parameters.

(v) produces the batch input program and optionally compiles it.

(vi) produces the batch output and formatting program and optionally compiles it.

(vii) produces the system and operating documentation and optionally formats it.

The batch totalling, if present, will not allow the operator to leave a batch on input until it balances or is deleted; the option notably missing from BDI (see section 1 of this chapter above).

The parameters are:

(i) file - name, title

(ii) record - name, maximum number of records, fixed or variable maximum

(iii) field - record name, field name, type, size, display name, range, valid values

(iv) record totalizers - record to be accumulated, record and field to check against

(v) field totalizers - field and record to be accumulated, add or subtract values, field and record to be checked against

Produced Batch Data Entry Program

This is a more complex version of the fm program. Note in particular the fewer lines of fixed code (again denoted by a vertical bar), about 50% less, and the individual record handling codings (the fm generator has only one record type and so the equivalent codings are static).

```
| IDENTIFICATION DIVISION.
  PROGRAM-ID. TIJOURNAL.
|*
|*  TPBDI - VER 2.0
|*
| ENVIRONMENT DIVISION.
| CONFIGURATION SECTION.
| SOURCE-COMPUTER. VAX-11.
| OBJECT-COMPUTER. VAX-11.
|*
| INPUT-OUTPUT SECTION.
| FILE-CONTROL.
|*
|* the file TP-FILE is the data entry file
|*
|     SELECT TP-FILE ASSIGN TO TPFILE
|         ORGANIZATION IS INDEXED
|         ACCESS IS DYNAMIC
|         RECORD KEY IS TP-KEY.
| DATA DIVISION.
| FILE SECTION.
| FD TP-FILE.
| 01 TP-KEY.
|     03 TP-BATCH-NUMBER          PIC 9999.
|     03 TP-RECORD-NUMBER         PIC 99.
|     03 TP-RECORD-TYPE           PIC X(10).
|     03 TP-LINE-NUMBER           PIC 9999.
  01  TP-HEADER-RECORD.
      03  FILLER                  PIC X(20).
      03  TP-HEADER-YEAR          PIC X.
      03  TP-HEADER-DATE          PIC 9(6).
      03  TP-HEADER-CREDIT        PIC 9(9)V9(2).
      03  TP-HEADER-DEBIT         PIC 9(9)V9(2).
      03  TP-HEADER-TYPE          PIC X.
  01  TP-CREDIT-RECORD.
      03  FILLER                  PIC X(20).
      03  TP-CREDIT-AMOUNT        PIC 9(9)V9(2).
      03  TP-CREDIT-ACCOUNT       PIC X(7).
      03  TP-CREDIT-ANALYSIS      PIC 9(2).
      03  TP-CREDIT-NARR          PIC X(50).
  01  TP-DEBIT-RECORD.
      03  FILLER                  PIC X(20).
      03  TP-DEBIT-AMOUNT         PIC 9(9)V9(2).
```

```
           03  TP-DEBIT-ACCOUNT          PIC X(7).
           03  TP-DEBIT-ANALYSIS         PIC 9(2).
           03  TP-DEBIT-NARR             PIC X(50).
       WORKING-STORAGE SECTION.
       01  FLAG-RECORD          PIC X.
       01  CHANGE-FLAG          PIC X.
           88  RECORD-CHANGED VALUE "C".
       01  HYPHENS              PIC X(80) VALUE SPACES.
       01  LAST-KEY-READ        PIC X(20).
       01  ROW                  PIC 99.
       01  COLUMN              PIC 99.
       01  REPLY                PIC X.
           88  NO-KEY VALUE "1".
           88  NO-RECORD VALUE "2".
           88  YES-RECORD VALUE "3".
           88  APPLY-CHANGE VALUE "4".
           88  TOTAL-FAIL VALUE "9".
       01  ERROR-CI.
           03  ECI-LIT          PIC X(20).
           03  ECI-STORE        PIC Z(17)9.
           03  FILLER           PIC X(6) VALUE " given".
           03  ECI-COUNT        PIC Z(17)9.
           03  FILLER           PIC X(12) VALUE " accumulated".
       01  ERROR-CR.
           03  ECR-LIT          PIC X(20).
           03  ECR-STORE        PIC -(12)9.9(5).
           03  FILLER           PIC X(6) VALUE " given".
           03  ECR-COUNT        PIC -(12)9.9(5).
           03  FILLER           PIC X(12) VALUE " accumulated".
       01  ST-FROM              PIC 99.
       01  ST-TO                PIC 99.
       01  SCROLL-FROM          PIC 99.
       01  SCROLL-TO            PIC 99.
       01  CURRENT-LINE              PIC 99 COMP.
       01  ST-SIZE             PIC 99 COMP.
       01  FILLER.
           03  INPUT-S            PIC X(60).
           88  NO-INPUT VALUE SPACES.
           03  IXX REDEFINES INPUT-S PIC X OCCURS 60.
      *
           03  OUTPUT-I         PIC -(17)9.
      *
           03  LOWER-I          PIC 9(18)-.
      *
           03  OUTPUT-R         PIC -(12)9.9(5).
      *
           03  UPPER-I          PIC 9(18)-.
       01  W-JULIAN-DATE-WORK-AREA.
           03  W-JD-DATE              PIC 9(6).
           03  FILLER REDEFINES W-JD-DATE.
               05  W-J-DAY            PIC 99.
               05  W-J-MONTH          PIC 99.
               05  W-J-YEAR           PIC 99.
           03  W-J-JULIAN             PIC 9(6) COMP.
       01  INPUT-D          PIC 9(6) COMP.
       01  LOWER-D          PIC 9(6) COMP.
       01  UPPER-D          PIC 9(6) COMP.
       01  SUBSCRIPT        PIC 999 COMP.
       01  WORK-1           PIC 99 COMP.
```

```
 01  WORK-2              PIC 99 COMP.
 01  ERROR-MESSAGE           PIC X(78).
 01  INPUT-I             PIC S9(18) COMP.
 01  LOWER-R             PIC S9(13)V9(5) COMP.
 01  UPPER-R             PIC S9(13)V9(5) COMP.
 01  INPUT-R             PIC S9(13)V9(5) COMP.
 01  DECIMALS            PIC 9.
 01  W-STORE-KEY.
     03  W-STORE-BATCH          PIC 9999.
     03  W-STORE-NUMBER         PIC 99.
     03  W-STORE-TYPE           PIC X(10).
     03  W-STORE-LINE           PIC 9999.
*
  01  W-COUNT-HEADER-CREDIT          PIC 9(9)V9(2) COMP.
  01  W-STORE-HEADER-CREDIT          PIC 9(9)V9(2) COMP.
  01  W-COUNT-HEADER-DEBIT           PIC 9(9)V9(2) COMP.
  01  W-STORE-HEADER-DEBIT           PIC 9(9)V9(2) COMP.
  01  W-HEADER                       PIC 9 COMP.
  01  W-CREDIT                       PIC 9(4) COMP.
  01  W-DEBIT                        PIC 9(4) COMP.
*
 PROCEDURE DIVISION.
 CONTROLX SECTION.
 C00.
     PERFORM INITIALX.
     PERFORM MAINFLOW.
     PERFORM CLOSEDOWN.
 C99.
     STOP RUN.
*
 INITIALX SECTION.
*
* open the files; clear the screen;
*   display the program title on
*   the vdu
*
 I00.
     OPEN I-O TP-FILE.
     MOVE 1 TO ROW.
     CALL "CLEARVDU" USING ROW.
     DISPLAY
     "Journal, Salary, Wage"
         WITH NO ADVANCING.
     MOVE 2 TO ROW.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "Batch number" WITH NO ADVANCING.
     INSPECT HYPHENS REPLACING ALL SPACES BY "-".
     MOVE 3 TO ROW.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY HYPHENS WITH NO ADVANCING.
 IN99.
     EXIT.
*
*
 MAINFLOW SECTION.
 M00.
     PERFORM GET-BATCH-NUMBER.
     IF NO-KEY
```

```
        GO TO M99.
    MOVE 23 TO ROW.
    MOVE 1 TO COLUMN.
    CALL "CURSORTO" USING ROW, COLUMN.
    IF YES-RECORD
        DISPLAY "Batch exists - continue? (Y=Yes)"
        WITH NO ADVANCING
        MOVE 34 TO COLUMN
    ELSE
        DISPLAY "Batch does not exist - continue? (Y=Yes)"
        WITH NO ADVANCING
        MOVE 42 TO COLUMN.
    MOVE 1 TO ST-SIZE.
    CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE.
    CALL "CLEARVDU" USING ROW.
    IF INPUT-S = "Y" OR INPUT-S = "y"
        IF YES-RECORD
            GO TO M05
        ELSE
            GO TO M02.
    GO TO M00.
M02.
    PERFORM ZEROIZE-TOTALS.
    MOVE W-STORE-BATCH TO TP-BATCH-NUMBER.
    MOVE SPACES TO TP-RECORD-TYPE.
    MOVE ZERO TO TP-LINE-NUMBER TP-RECORD-NUMBER.
    WRITE TP-KEY INVALID KEY
        MOVE "FAILED TO WRITE FIRST RECORD" TO ERROR-MESSAGE
        GO TO FAIL-END.
    PERFORM INSERT-HEADER-R.
    PERFORM INSERT-CREDIT-R.
    PERFORM INSERT-DEBIT-R.
    GO TO M15.
M05.
    MOVE 23 TO ROW.
    MOVE 1 TO COLUMN.
    CALL "CURSORTO" USING ROW, COLUMN.
    DISPLAY "Amend or Delete this batch (D=Delete)"
       WITH NO ADVANCING.
    MOVE 1 TO ST-SIZE.
    MOVE 40 TO COLUMN.
    CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE.
    CALL "CLEARVDU" USING ROW.
    IF INPUT-S NOT = "D"
        GO TO M10.
    PERFORM INQUIRE-OK.
    IF APPLY-CHANGE
        PERFORM DELETE-BATCH
        GO TO M00.
    GO TO M05.
M10.
    PERFORM READ-START-BATCH.
    IF NO-RECORD
        MOVE "FAILED TO READ FIRST RECORD" TO ERROR-MESSAGE
        GO TO FAIL-END.
    PERFORM ZEROIZE-TOTALS.
    PERFORM READ-NEXT-TP.
    PERFORM AMEND-HEADER-R.
    PERFORM AMEND-CREDIT-R.
```

```
        PERFORM AMEND-DEBIT-R.
    M15.
        MOVE ZERO TO REPLY.
        IF W-STORE-HEADER-CREDIT NOT = W-COUNT-HEADER-CREDIT
            MOVE "credit count wrong" TO ECR-LIT
            MOVE W-STORE-HEADER-CREDIT TO ECR-STORE
            MOVE W-COUNT-HEADER-CREDIT TO ECR-COUNT
            MOVE ERROR-CR TO ERROR-MESSAGE
            MOVE 9 TO REPLY
            CALL "ERRORVDU" USING ROW, COLUMN,
            ST-SIZE, ERROR-MESSAGE, INPUT-S.
        IF W-STORE-HEADER-DEBIT NOT = W-COUNT-HEADER-DEBIT
            MOVE "debit count wrong" TO ECR-LIT
            MOVE W-STORE-HEADER-DEBIT TO ECR-STORE
            MOVE W-COUNT-HEADER-DEBIT TO ECR-COUNT
            MOVE ERROR-CR TO ERROR-MESSAGE
            MOVE 9 TO REPLY
            CALL "ERRORVDU" USING ROW, COLUMN,
            ST-SIZE, ERROR-MESSAGE, INPUT-S.
        IF TOTAL-FAIL
            GO TO M05.
        MOVE "Batch totals correct" TO ERROR-MESSAGE.
        CALL "ERRORVDU" USING ROW, COLUMN,
            ST-SIZE, ERROR-MESSAGE, INPUT-S.
        GO TO M00.
    M99.
        EXIT.
    *
    * close files before the end of the
    * run; set scrolling back to lines 1 to 24
    *
    CLOSEDOWN SECTION.
    C00.
        MOVE 1 TO SCROLL-FROM.
        MOVE 24 TO SCROLL-TO.
        CALL "SCROLL" USING SCROLL-FROM, SCROLL-TO.
        CLOSE TP-FILE.
    C99.
        EXIT.
    *
    * read the first record in a batch
    *
    READ-START-BATCH SECTION.
    RSB00.
        MOVE W-STORE-BATCH TO TP-BATCH-NUMBER.
        MOVE ZERO TO TP-LINE-NUMBER TP-RECORD-NUMBER.
        MOVE SPACES TO TP-RECORD-TYPE.
        PERFORM READ-TP.
    RSB99.
        EXIT.
    *
    * read the (next) record on the work file
    *
    READ-TP SECTION.
    RW00.
        MOVE 3 TO REPLY.
        READ TP-FILE INVALID KEY
            MOVE 2 TO REPLY.
    RW99.
```

```
        EXIT.
*
 READ-NEXT-TP SECTION.
 RN00.
        MOVE 3 TO REPLY.
        READ TP-FILE NEXT AT END
            MOVE 2 TO REPLY.
 RN99.
        EXIT.
*
* accept the batch number from the screen then check the
*    file that the record is absent (for insert)
*    or present (for amend).
*
 GET-BATCH-NUMBER SECTION.
 GE00.
        MOVE 4 TO ROW.
        CALL "CLEARVDU" USING ROW.
        MOVE ZERO TO LOWER-I.
        MOVE 9999 TO UPPER-I.
        MOVE 2 TO ROW.
        MOVE 14 TO COLUMN.
        MOVE 44 TO ST-SIZE.
        CALL "CLEARAREA" USING ROW, COLUMN, ST-SIZE.
        CALL "INPUTI" USING ROW, COLUMN, LOWER-I, UPPER-I,
            INPUT-I.
        IF INPUT-I = ZERO
            MOVE 1 TO REPLY
            GO TO GE99.
        MOVE INPUT-I TO W-STORE-BATCH.
        PERFORM READ-START-BATCH.
        MOVE 21 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        IF NO-RECORD
            DISPLAY "Insert batch" WITH NO ADVANCING
        ELSE
            DISPLAY "Amend batch " WITH NO ADVANCING.
 GE99.
        EXIT.
*
* zeroize totals accummulators
*
 ZEROIZE-TOTALS SECTION.
 ZE00.
        MOVE ZERO TO W-COUNT-HEADER-CREDIT.
        MOVE ZERO TO W-COUNT-HEADER-DEBIT.
 ZE99.
        EXIT.
*
* these control inputting new records onto the file. They
* allows initial input of fields, amending of any errors,
* the option to abandon the insertion, writing of the
* record to the file.
*
*
*
 INSERT-HEADER-R SECTION.
 IN-HEADER-00.
        MOVE "HEADER     " TO TP-RECORD-TYPE.
```

```
           MOVE 1 TO TP-RECORD-NUMBER.
           MOVE 41 TO COLUMN.
           MOVE 2 TO ROW.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY "Insert header     " WITH NO ADVANCING.
           MOVE 1 TO TP-LINE-NUMBER.
           MOVE ZERO TO W-HEADER.
           PERFORM NEW-HEADER-PAGE.
       IN-HEADER-05.
           PERFORM INPUT-HEADER-FIELDS.
           PERFORM AMEND-HEADER-FIELDS.
           WRITE TP-HEADER-RECORD INVALID KEY
               MOVE "Write FAIL" TO ERROR-MESSAGE
               GO TO FAIL-END.
           ADD 1 TO W-HEADER.
           MOVE TP-HEADER-CREDIT TO W-STORE-HEADER-CREDIT.
           MOVE TP-HEADER-DEBIT TO W-STORE-HEADER-DEBIT.
           IF W-HEADER < 1
               ADD 1 TO TP-LINE-NUMBER
               GO TO IN-HEADER-05.
       IN-HEADER-99.
           EXIT.
      *
       INSERT-CREDIT-R SECTION.
       IN-CREDIT-00.
           MOVE "CREDIT     " TO TP-RECORD-TYPE.
           MOVE 2 TO TP-RECORD-NUMBER.
           MOVE 41 TO COLUMN.
           MOVE 2 TO ROW.
           CALL "CURSORTO" USING ROW, COLUMN.
           DISPLAY "Insert credit     " WITH NO ADVANCING.
           MOVE 1 TO TP-LINE-NUMBER.
           MOVE ZERO TO W-CREDIT.
           PERFORM NEW-CREDIT-PAGE.
       IN-CREDIT-05.
           PERFORM INPUT-CREDIT-FIELDS.
           IF FLAG-RECORD = "N"
               GO TO IN-CREDIT-99.
           PERFORM AMEND-CREDIT-FIELDS.
           IF FLAG-RECORD = "N"
               GO TO IN-CREDIT-99.
           WRITE TP-CREDIT-RECORD INVALID KEY
               MOVE "Write FAIL" TO ERROR-MESSAGE
               GO TO FAIL-END.
           ADD 1 TO W-CREDIT.
           ADD TP-CREDIT-AMOUNT TO W-COUNT-HEADER-CREDIT.
           IF W-CREDIT < 9999
               ADD 1 TO TP-LINE-NUMBER
               GO TO IN-CREDIT-05.
       IN-CREDIT-99.
           EXIT.
      *
       INSERT-DEBIT-R SECTION.
       IN-DEBIT-00.
           MOVE "DEBIT      " TO TP-RECORD-TYPE.
           MOVE 3 TO TP-RECORD-NUMBER.
           MOVE 41 TO COLUMN.
           MOVE 2 TO ROW.
           CALL "CURSORTO" USING ROW, COLUMN.
```

```
        DISPLAY "Insert debit      " WITH NO ADVANCING.
        MOVE 1 TO TP-LINE-NUMBER.
        MOVE ZERO TO W-DEBIT.
        PERFORM NEW-DEBIT-PAGE.
    IN-DEBIT-05.
        PERFORM INPUT-DEBIT-FIELDS.
        IF FLAG-RECORD = "N"
            GO TO IN-DEBIT-99.
        PERFORM AMEND-DEBIT-FIELDS.
        IF FLAG-RECORD = "N"
            GO TO IN-DEBIT-99.
        WRITE TP-DEBIT-RECORD INVALID KEY
            MOVE "Write FAIL" TO ERROR-MESSAGE
            GO TO FAIL-END.
        ADD 1 TO W-DEBIT.
        ADD TP-DEBIT-AMOUNT TO W-COUNT-HEADER-DEBIT.
        IF W-DEBIT < 9999
            ADD 1 TO TP-LINE-NUMBER
            GO TO IN-DEBIT-05.
    IN-DEBIT-99.
        EXIT.
*
*  these control the amending of existing records. They
*   displays the details, allow amending of
*   specific field values, give the option to abandon
*   the amendments, give the option to delete the record
*   or write the new record values.
*
*
*
    AMEND-HEADER-R SECTION.
    AM-HEADER-00.
        MOVE 41 TO COLUMN.
        MOVE 2 TO ROW.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "Amend header    " WITH NO ADVANCING.
        PERFORM NEW-HEADER-PAGE.
        MOVE ZERO TO W-HEADER.
    AM-HEADER-05.
        PERFORM SHOW-HEADER-FIELDS.
        MOVE SPACE TO CHANGE-FLAG.
        PERFORM AMEND-HEADER-FIELDS.
        IF NOT RECORD-CHANGED
            GO TO AM-HEADER-15.
        PERFORM INQUIRE-OK.
        IF APPLY-CHANGE
            GO TO AM-HEADER-10.
        GO TO AM-HEADER-15.
    AM-HEADER-10.
        REWRITE TP-HEADER-RECORD INVALID KEY
            MOVE "Rewrite fail" TO ERROR-MESSAGE
            GO TO FAIL-END.
    AM-HEADER-15.
        ADD 1 TO W-HEADER.
        MOVE TP-HEADER-CREDIT TO W-STORE-HEADER-CREDIT.
        MOVE TP-HEADER-DEBIT TO W-STORE-HEADER-DEBIT.
        MOVE TP-KEY TO W-STORE-KEY.
        PERFORM READ-NEXT-TP.
        IF YES-RECORD
```

```
        AND TP-BATCH-NUMBER = W-STORE-BATCH
        AND TP-RECORD-TYPE = W-STORE-TYPE
            GO TO AM-HEADER-05.
    AM-HEADER-99.
        EXIT.
*
    AMEND-CREDIT-R SECTION.
    AM-CREDIT-00.
        MOVE 41 TO COLUMN.
        MOVE 2 TO ROW.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "Amend credit    " WITH NO ADVANCING.
        PERFORM NEW-CREDIT-PAGE.
        MOVE ZERO TO W-CREDIT.
        IF TP-RECORD-TYPE NOT = "CREDIT     "
            MOVE ZERO TO W-STORE-LINE
            GO TO AM-CREDIT-20.
    AM-CREDIT-05.
        PERFORM SHOW-CREDIT-FIELDS.
        MOVE SPACE TO CHANGE-FLAG.
        PERFORM AMEND-CREDIT-FIELDS.
        IF NOT RECORD-CHANGED
            GO TO AM-CREDIT-06.
        PERFORM INQUIRE-OK.
        IF APPLY-CHANGE
            GO TO AM-CREDIT-10.
    AM-CREDIT-06.
        MOVE 23 TO ROW.
        MOVE 1 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "Retain or Delete this record (D=Delete)"
            WITH NO ADVANCING.
        MOVE 1 TO ST-SIZE.
        MOVE 41 TO COLUMN.
        CALL "INPUTS" USING ROW, COLUMN,
        INPUT-S, ST-SIZE.
        CALL "CLEARVDU" USING ROW.
        IF INPUT-S NOT = "D"
            GO TO AM-CREDIT-11.
        DELETE TP-FILE RECORD INVALID KEY
            MOVE "Delete record fail" TO ERROR-MESSAGE
            GO TO FAIL-END.
        MOVE 1 TO COLUMN.
        SUBTRACT 1 FROM CURRENT-LINE GIVING ROW.
        MOVE 79 TO ST-SIZE.
    AM-CREDIT-09.
        CALL "CLEARAREA" USING ROW, COLUMN, ST-SIZE.
        IF ROW NOT > CURRENT-LINE
            ADD 1 TO ROW
            GO TO AM-CREDIT-09.
        SUBTRACT 2 FROM CURRENT-LINE.
        GO TO AM-CREDIT-15.
    AM-CREDIT-10.
        REWRITE TP-CREDIT-RECORD INVALID KEY
            MOVE "Rewrite fail" TO ERROR-MESSAGE
            GO TO FAIL-END.
    AM-CREDIT-11.
        ADD 1 TO W-CREDIT.
        ADD TP-CREDIT-AMOUNT TO W-COUNT-HEADER-CREDIT.
```

```
AM-CREDIT-15.
    MOVE TP-KEY TO W-STORE-KEY.
    PERFORM READ-NEXT-TP.
    IF YES-RECORD
    AND TP-BATCH-NUMBER = W-STORE-BATCH
    AND TP-RECORD-TYPE = W-STORE-TYPE
        GO TO AM-CREDIT-05.
    IF YES-RECORD
        MOVE TP-KEY TO LAST-KEY-READ
    ELSE
        MOVE SPACES TO LAST-KEY-READ.
    IF W-CREDIT = 9999
        GO TO AM-CREDIT-25.
AM-CREDIT-20.
    MOVE 23 TO ROW.
    MOVE 1 TO COLUMN.
    CALL "CURSORTO" USING ROW, COLUMN.
    DISPLAY "Add more records (N=No)" WITH NO ADVANCING.
    MOVE 27 TO COLUMN.
    MOVE 1 TO ST-SIZE.
    CALL "INPUTS" USING ROW, COLUMN,
    INPUT-S, ST-SIZE.
    CALL "CLEARVDU" USING ROW.
    IF INPUT-S = "N"
        GO TO AM-CREDIT-25.
AM-CREDIT-21.
    ADD 1 W-STORE-LINE GIVING TP-LINE-NUMBER.
    MOVE "CREDIT     " TO TP-RECORD-TYPE.
    MOVE 2 TO TP-RECORD-NUMBER.
    MOVE W-STORE-BATCH TO TP-BATCH-NUMBER.
    PERFORM INPUT-CREDIT-FIELDS.
    IF FLAG-RECORD = "N"
        SUBTRACT 1 FROM CURRENT-LINE
        GO TO AM-CREDIT-20.
    PERFORM AMEND-CREDIT-FIELDS.
    IF FLAG-RECORD = "N"
        SUBTRACT 1 FROM CURRENT-LINE
        GO TO AM-CREDIT-20.
    MOVE SPACE TO CHANGE-FLAG.
    WRITE TP-CREDIT-RECORD INVALID KEY
        MOVE "Write FAIL" TO ERROR-MESSAGE
        GO TO FAIL-END.
AM-CREDIT-22.
    ADD 1 TO W-STORE-LINE.
    ADD 1 TO W-CREDIT.
    ADD TP-CREDIT-AMOUNT TO W-COUNT-HEADER-CREDIT.
    IF W-CREDIT NOT = 9999
        GO TO AM-CREDIT-21.
AM-CREDIT-25.
    MOVE LAST-KEY-READ TO TP-KEY.
    IF LAST-KEY-READ = SPACES
        GO TO AM-CREDIT-99.
    PERFORM READ-TP.
    IF NO-RECORD
        MOVE "Reread FAIL" TO ERROR-MESSAGE
        GO TO FAIL-END.
AM-CREDIT-99.
    EXIT.
*
```

```
AMEND-DEBIT-R SECTION.
AM-DEBIT-00.
    MOVE 41 TO COLUMN.
    MOVE 2 TO ROW.
    CALL "CURSORTO" USING ROW, COLUMN.
    DISPLAY "Amend debit     " WITH NO ADVANCING.
    PERFORM NEW-DEBIT-PAGE.
    MOVE ZERO TO W-DEBIT.
    IF TP-RECORD-TYPE NOT = "DEBIT      "
        MOVE ZERO TO W-STORE-LINE
        GO TO AM-DEBIT-20.
AM-DEBIT-05.
    PERFORM SHOW-DEBIT-FIELDS.
    MOVE SPACE TO CHANGE-FLAG.
    PERFORM AMEND-DEBIT-FIELDS.
    IF NOT RECORD-CHANGED
        GO TO AM-DEBIT-06.
    PERFORM INQUIRE-OK.
    IF APPLY-CHANGE
        GO TO AM-DEBIT-10.
AM-DEBIT-06.
    MOVE 23 TO ROW.
    MOVE 1 TO COLUMN.
    CALL "CURSORTO" USING ROW, COLUMN.
    DISPLAY "Retain or Delete this record (D=Delete)"
        WITH NO ADVANCING.
    MOVE 1 TO ST-SIZE.
    MOVE 41 TO COLUMN.
    CALL "INPUTS" USING ROW, COLUMN,
    INPUT-S, ST-SIZE.
    CALL "CLEARVDU" USING ROW.
    IF INPUT-S NOT = "D"
        GO TO AM-DEBIT-11.
    DELETE TP-FILE RECORD INVALID KEY
        MOVE "Delete record fail" TO ERROR-MESSAGE
        GO TO FAIL-END.
    MOVE 1 TO COLUMN.
    SUBTRACT 1 FROM CURRENT-LINE GIVING ROW.
    MOVE 79 TO ST-SIZE.
AM-DEBIT-09.
    CALL "CLEARAREA" USING ROW, COLUMN, ST-SIZE.
    IF ROW NOT > CURRENT-LINE
        ADD 1 TO ROW
        GO TO AM-DEBIT-09.
    SUBTRACT 2 FROM CURRENT-LINE.
    GO TO AM-DEBIT-15.
AM-DEBIT-10.
    REWRITE TP-DEBIT-RECORD INVALID KEY
        MOVE "Rewrite fail" TO ERROR-MESSAGE
        GO TO FAIL-END.
AM-DEBIT-11.
    ADD 1 TO W-DEBIT.
    ADD TP-DEBIT-AMOUNT TO W-COUNT-HEADER-DEBIT.
AM-DEBIT-15.
    MOVE TP-KEY TO W-STORE-KEY.
    PERFORM READ-NEXT-TP.
    IF YES-RECORD
    AND TP-BATCH-NUMBER = W-STORE-BATCH
    AND TP-RECORD-TYPE = W-STORE-TYPE
```

```
            GO TO AM-DEBIT-05.
        IF YES-RECORD
            MOVE TP-KEY TO LAST-KEY-READ
        ELSE
            MOVE SPACES TO LAST-KEY-READ.
        IF W-DEBIT = 9999
            GO TO AM-DEBIT-25.
    AM-DEBIT-20.
        MOVE 23 TO ROW.
        MOVE 1 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "Add more records (N=No)" WITH NO ADVANCING.
        MOVE 27 TO COLUMN.
        MOVE 1 TO ST-SIZE.
        CALL "INPUTS" USING ROW, COLUMN,
        INPUT-S, ST-SIZE.
        CALL "CLEARVDU" USING ROW.
        IF INPUT-S = "N"
            GO TO AM-DEBIT-25.
    AM-DEBIT-21.
        ADD 1 W-STORE-LINE GIVING TP-LINE-NUMBER.
        MOVE "DEBIT      " TO TP-RECORD-TYPE.
        MOVE 3 TO TP-RECORD-NUMBER.
        MOVE W-STORE-BATCH TO TP-BATCH-NUMBER.
        PERFORM INPUT-DEBIT-FIELDS.
        IF FLAG-RECORD = "N"
            SUBTRACT 1 FROM CURRENT-LINE
            GO TO AM-DEBIT-20.
        PERFORM AMEND-DEBIT-FIELDS.
        IF FLAG-RECORD = "N"
            SUBTRACT 1 FROM CURRENT-LINE
            GO TO AM-DEBIT-20.
        MOVE SPACE TO CHANGE-FLAG.
        WRITE TP-DEBIT-RECORD INVALID KEY
            MOVE "Write FAIL" TO ERROR-MESSAGE
            GO TO FAIL-END.
    AM-DEBIT-22.
        ADD 1 TO W-STORE-LINE.
        ADD 1 TO W-DEBIT.
        ADD TP-DEBIT-AMOUNT TO W-COUNT-HEADER-DEBIT.
        IF W-DEBIT NOT = 9999
            GO TO AM-DEBIT-21.
    AM-DEBIT-25.
        MOVE LAST-KEY-READ TO TP-KEY.
        IF LAST-KEY-READ = SPACES
            GO TO AM-DEBIT-99.
        PERFORM READ-TP.
        IF NO-RECORD
            MOVE "Reread FAIL" TO ERROR-MESSAGE
            GO TO FAIL-END.
    AM-DEBIT-99.
        EXIT.
*
* this section deletes a whole batch
*
 DELETE-BATCH SECTION.
 DB00.
        MOVE W-STORE-BATCH TO TP-BATCH-NUMBER.
        MOVE SPACES TO TP-RECORD-TYPE.
```

```
        MOVE ZERO TO TP-LINE-NUMBER TP-RECORD-NUMBER.
        PERFORM READ-TP.
    DB05.
        IF NO-RECORD OR W-STORE-BATCH NOT = TP-BATCH-NUMBER
            GO TO DB10.
        DELETE TP-FILE RECORD INVALID KEY
            MOVE "DELETE FAIL" TO ERROR-MESSAGE
            GO TO FAIL-END.
        PERFORM READ-NEXT-TP.
        GO TO DB05.
    DB10.
        MOVE "Batch Deleted" TO ERROR-MESSAGE.
        MOVE 23 TO ROW.
        CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE,
            ERROR-MESSAGE, INPUT-S.
    DB99.
        EXIT.
*
* these sections control global inputting of
*   all fields
*
*
*
    INPUT-HEADER-FIELDS SECTION.
    INP-HEADER-00.
        IF CURRENT-LINE < 21
            ADD 1 TO CURRENT-LINE
        ELSE
            PERFORM SCROLL-AREA.
        MOVE CURRENT-LINE TO ROW.
        PERFORM INPUT-HEADER-YEAR.
        PERFORM INPUT-HEADER-DATE.
        PERFORM INPUT-HEADER-CREDIT.
        PERFORM INPUT-HEADER-DEBIT.
        PERFORM INPUT-HEADER-TYPE.
    INP-HEADER-99.
        EXIT.
*
    INPUT-CREDIT-FIELDS SECTION.
    INP-CREDIT-00.
        IF CURRENT-LINE < 21
            ADD 1 TO CURRENT-LINE
        ELSE
            PERFORM SCROLL-AREA.
        MOVE CURRENT-LINE TO ROW.
        PERFORM INPUT-CREDIT-AMOUNT.
        IF NO-INPUT
            GO TO INP-CREDIT-99.
        PERFORM INPUT-CREDIT-ACCOUNT.
        PERFORM INPUT-CREDIT-ANALYSIS.
        IF CURRENT-LINE < 21
            ADD 1 TO CURRENT-LINE
        ELSE
            PERFORM SCROLL-AREA.
        MOVE CURRENT-LINE TO ROW.
        PERFORM INPUT-CREDIT-NARR.
    INP-CREDIT-99.
        EXIT.
*
```

```
 INPUT-DEBIT-FIELDS SECTION.
 INP-DEBIT-00.
     IF CURRENT-LINE < 21
         ADD 1 TO CURRENT-LINE
     ELSE
         PERFORM SCROLL-AREA.
     MOVE CURRENT-LINE TO ROW.
     PERFORM INPUT-DEBIT-AMOUNT.
     IF NO-INPUT
         GO TO INP-DEBIT-99.
     PERFORM INPUT-DEBIT-ACCOUNT.
     PERFORM INPUT-DEBIT-ANALYSIS.
     IF CURRENT-LINE < 21
         ADD 1 TO CURRENT-LINE
     ELSE
         PERFORM SCROLL-AREA.
     MOVE CURRENT-LINE TO ROW.
     PERFORM INPUT-DEBIT-NARR.
 INP-DEBIT-99.
     EXIT.
*
* these functions control selection of amendment
*   of an individual field
*
*
*
 AMEND-HEADER-FIELDS SECTION.
 AME-HEADER-00.
     MOVE 23 TO ROW.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "Which field" WITH NO ADVANCING.
     MOVE 14 TO COLUMN.
     MOVE 10 TO ST-SIZE.
     CALL "INPUTS" USING ROW, COLUMN,
     INPUT-S, ST-SIZE.
     CALL "CLEARVDU" USING ROW.
     IF INPUT-S = SPACES
         GO TO AME-HEADER-99.
     IF INPUT-S = "year"
         MOVE "C" TO CHANGE-FLAG
         MOVE CURRENT-LINE TO ROW
         PERFORM INPUT-HEADER-YEAR
     ELSE
     IF INPUT-S = "date"
         MOVE "C" TO CHANGE-FLAG
         MOVE CURRENT-LINE TO ROW
         PERFORM INPUT-HEADER-DATE
     ELSE
     IF INPUT-S = "credit"
         MOVE "C" TO CHANGE-FLAG
         MOVE CURRENT-LINE TO ROW
         PERFORM INPUT-HEADER-CREDIT
     ELSE
     IF INPUT-S = "debit"
         MOVE "C" TO CHANGE-FLAG
         MOVE CURRENT-LINE TO ROW
         PERFORM INPUT-HEADER-DEBIT
     ELSE
```

```
     IF INPUT-S = "type"
         MOVE "C" TO CHANGE-FLAG
         MOVE CURRENT-LINE TO ROW
         PERFORM INPUT-HEADER-TYPE
     ELSE
         MOVE
     "One of: year, date, credit, debit, type"
         TO ERROR-MESSAGE
         CALL "ERRORVDU" USING ROW, COLUMN,
         ST-SIZE, ERROR-MESSAGE, INPUT-S.
     GO TO AME-HEADER-00.
 AME-HEADER-99.
     EXIT.
*
 AMEND-CREDIT-FIELDS SECTION.
 AME-CREDIT-00.
     MOVE 23 TO ROW.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "Which field" WITH NO ADVANCING.
     MOVE 14 TO COLUMN.
     MOVE 10 TO ST-SIZE.
     CALL "INPUTS" USING ROW, COLUMN,
     INPUT-S, ST-SIZE.
     CALL "CLEARVDU" USING ROW.
     IF INPUT-S = SPACES
         GO TO AME-CREDIT-99.
     IF INPUT-S = "amount"
         MOVE "C" TO CHANGE-FLAG
         SUBTRACT 1 FROM CURRENT-LINE GIVING ROW
         PERFORM INPUT-CREDIT-AMOUNT
     ELSE
     IF INPUT-S = "account"
         MOVE "C" TO CHANGE-FLAG
         SUBTRACT 1 FROM CURRENT-LINE GIVING ROW
         PERFORM INPUT-CREDIT-ACCOUNT
     ELSE
     IF INPUT-S = "analysis"
         MOVE "C" TO CHANGE-FLAG
         SUBTRACT 1 FROM CURRENT-LINE GIVING ROW
         PERFORM INPUT-CREDIT-ANALYSIS
     ELSE
     IF INPUT-S = "narrative"
         MOVE "C" TO CHANGE-FLAG
         MOVE CURRENT-LINE TO ROW
         PERFORM INPUT-CREDIT-NARR
     ELSE
         MOVE
     "One of: amount, account, analysis, narrative"
         TO ERROR-MESSAGE
         CALL "ERRORVDU" USING ROW, COLUMN,
         ST-SIZE, ERROR-MESSAGE, INPUT-S.
     GO TO AME-CREDIT-00.
 AME-CREDIT-99.
     EXIT.
*
 AMEND-DEBIT-FIELDS SECTION.
 AME-DEBIT-00.
     MOVE 23 TO ROW.
```

```
      MOVE 1 TO COLUMN.
      CALL "CURSORTO" USING ROW, COLUMN.
      DISPLAY "Which field" WITH NO ADVANCING.
      MOVE 14 TO COLUMN.
      MOVE 10 TO ST-SIZE.
      CALL "INPUTS" USING ROW, COLUMN,
      INPUT-S, ST-SIZE.
      CALL "CLEARVDU" USING ROW.
      IF INPUT-S = SPACES
          GO TO AME-DEBIT-99.
      IF INPUT-S = "amount"
          MOVE "C" TO CHANGE-FLAG
          SUBTRACT 1 FROM CURRENT-LINE GIVING ROW
          PERFORM INPUT-DEBIT-AMOUNT
      ELSE
      IF INPUT-S = "account"
          MOVE "C" TO CHANGE-FLAG
          SUBTRACT 1 FROM CURRENT-LINE GIVING ROW
          PERFORM INPUT-DEBIT-ACCOUNT
      ELSE
      IF INPUT-S = "analysis"
          MOVE "C" TO CHANGE-FLAG
          SUBTRACT 1 FROM CURRENT-LINE GIVING ROW
          PERFORM INPUT-DEBIT-ANALYSIS
      ELSE
      IF INPUT-S = "narrative"
          MOVE "C" TO CHANGE-FLAG
          MOVE CURRENT-LINE TO ROW
          PERFORM INPUT-DEBIT-NARR
      ELSE
          MOVE
      "One of: amount, account, analysis, narrative"
          TO ERROR-MESSAGE
          CALL "ERRORVDU" USING ROW, COLUMN,
          ST-SIZE, ERROR-MESSAGE, INPUT-S.
      GO TO AME-DEBIT-00.
  AME-DEBIT-99.
      EXIT.
*
* these functions allow the display of the
*   fields on a record
*
*
*
  SHOW-HEADER-FIELDS SECTION.
  SHO-HEADER-00.
      IF CURRENT-LINE < 21
          ADD 1 TO CURRENT-LINE
      ELSE
          PERFORM SCROLL-AREA.
      MOVE CURRENT-LINE TO ROW.
      PERFORM DISPLAY-HEADER-YEAR.
      PERFORM DISPLAY-HEADER-DATE.
      PERFORM DISPLAY-HEADER-CREDIT.
      PERFORM DISPLAY-HEADER-DEBIT.
      PERFORM DISPLAY-HEADER-TYPE.
  SHO-HEADER-99.
      EXIT.
*
```

```
 SHOW-CREDIT-FIELDS SECTION.
 SHO-CREDIT-00.
     IF CURRENT-LINE < 21
         ADD 1 TO CURRENT-LINE
     ELSE
         PERFORM SCROLL-AREA.
     MOVE CURRENT-LINE TO ROW.
     PERFORM DISPLAY-CREDIT-AMOUNT.
     PERFORM DISPLAY-CREDIT-ACCOUNT.
     PERFORM DISPLAY-CREDIT-ANALYSIS.
     IF CURRENT-LINE < 21
         ADD 1 TO CURRENT-LINE
     ELSE
         PERFORM SCROLL-AREA.
     MOVE CURRENT-LINE TO ROW.
     PERFORM DISPLAY-CREDIT-NARR.
 SHO-CREDIT-99.
     EXIT.
*
 SHOW-DEBIT-FIELDS SECTION.
 SHO-DEBIT-00.
     IF CURRENT-LINE < 21
         ADD 1 TO CURRENT-LINE
     ELSE
         PERFORM SCROLL-AREA.
     MOVE CURRENT-LINE TO ROW.
     PERFORM DISPLAY-DEBIT-AMOUNT.
     PERFORM DISPLAY-DEBIT-ACCOUNT.
     PERFORM DISPLAY-DEBIT-ANALYSIS.
     IF CURRENT-LINE < 21
         ADD 1 TO CURRENT-LINE
     ELSE
         PERFORM SCROLL-AREA.
     MOVE CURRENT-LINE TO ROW.
     PERFORM DISPLAY-DEBIT-NARR.
 SHO-DEBIT-99.
     EXIT.
*
* the following sections control the inputting
*   and verification of the individual fields
*   on the TP work file.
*
*
*
 INPUT-HEADER-YEAR SECTION.
 I-HEADER-YEAR-00.
     MOVE 1 TO COLUMN.
 I-HEADER-YEAR-05.
     MOVE 1 TO ST-SIZE.
     CALL "INPUTS" USING ROW, COLUMN,
     INPUT-S, ST-SIZE.
     MOVE INPUT-S TO TP-HEADER-YEAR.
     IF TP-HEADER-YEAR = "O" OR "C"
         GO TO I-HEADER-YEAR-99.
     MOVE "One of  O  C  " TO ERROR-MESSAGE.
         CALL "ERRORVDU" USING ROW, COLUMN,
         ST-SIZE, ERROR-MESSAGE, INPUT-S.
     GO TO I-HEADER-YEAR-05.
 I-HEADER-YEAR-99.
```

```
     EXIT.
*
 INPUT-HEADER-DATE SECTION.
 I-HEADER-DATE-00.
     MOVE 6 TO COLUMN.
     MOVE ZERO TO LOWER-D.
     MOVE 311249 TO UPPER-D .
 I-HEADER-DATE-05.
     CALL "INPUTD" USING ROW, COLUMN,
     LOWER-D, UPPER-D, INPUT-D.
     MOVE INPUT-D TO TP-HEADER-DATE.
 I-HEADER-DATE-99.
     EXIT.
*
 INPUT-HEADER-CREDIT SECTION.
 I-HEADER-CREDIT-00.
     MOVE 13 TO COLUMN.
     MOVE ZERO TO LOWER-R.
     MOVE 999999999.99 TO UPPER-R .
     MOVE 2 TO DECIMALS.
 I-HEADER-CREDIT-05.
     CALL "INPUTR" USING ROW, COLUMN,
     LOWER-R, UPPER-R, INPUT-R, DECIMALS.
     MOVE INPUT-R TO TP-HEADER-CREDIT.
 I-HEADER-CREDIT-99.
     EXIT.
*
 INPUT-HEADER-DEBIT SECTION.
 I-HEADER-DEBIT-00.
     MOVE 26 TO COLUMN.
     MOVE ZERO TO LOWER-R.
     MOVE 999999999.99 TO UPPER-R .
     MOVE 2 TO DECIMALS.
 I-HEADER-DEBIT-05.
     CALL "INPUTR" USING ROW, COLUMN,
     LOWER-R, UPPER-R, INPUT-R, DECIMALS.
     MOVE INPUT-R TO TP-HEADER-DEBIT.
 I-HEADER-DEBIT-99.
     EXIT.
*
 INPUT-HEADER-TYPE SECTION.
 I-HEADER-TYPE-00.
     MOVE 39 TO COLUMN.
 I-HEADER-TYPE-05.
     MOVE 1 TO ST-SIZE.
     CALL "INPUTS" USING ROW, COLUMN,
     INPUT-S, ST-SIZE.
     MOVE INPUT-S TO TP-HEADER-TYPE.
     IF TP-HEADER-TYPE = "J" OR "S" OR "W"
         GO TO I-HEADER-TYPE-99.
     MOVE "One of  J  S  W  " TO ERROR-MESSAGE.
         CALL "ERRORVDU" USING ROW, COLUMN,
         ST-SIZE, ERROR-MESSAGE, INPUT-S.
     GO TO I-HEADER-TYPE-05.
 I-HEADER-TYPE-99.
     EXIT.
*
 INPUT-CREDIT-AMOUNT SECTION.
 I-CREDIT-AMOUNT-00.
```

```
      MOVE 1 TO COLUMN.
      MOVE ZERO TO LOWER-R.
      MOVE 999999999.99 TO UPPER-R .
      MOVE 2 TO DECIMALS.
 I-CREDIT-AMOUNT-05.
      CALL "INPUTRT" USING ROW, COLUMN,
      LOWER-R, UPPER-R, INPUT-R, INPUT-S, DECIMALS.
      IF INPUT-S = SPACE
          MOVE "N" TO FLAG-RECORD
          GO TO I-CREDIT-AMOUNT-99.
      MOVE SPACE TO FLAG-RECORD.
      MOVE INPUT-R TO TP-CREDIT-AMOUNT.
 I-CREDIT-AMOUNT-99.
      EXIT.
*
 INPUT-CREDIT-ACCOUNT SECTION.
 I-CREDIT-ACCOUNT-00.
      MOVE 14 TO COLUMN.
 I-CREDIT-ACCOUNT-05.
      MOVE 7 TO ST-SIZE.
      CALL "INPUTS" USING ROW, COLUMN,
      INPUT-S, ST-SIZE.
      MOVE INPUT-S TO TP-CREDIT-ACCOUNT.
 I-CREDIT-ACCOUNT-99.
      EXIT.
*
 INPUT-CREDIT-ANALYSIS SECTION.
 I-CREDIT-ANALYSIS-00.
      MOVE 22 TO COLUMN.
      MOVE ZERO TO LOWER-I.
      MOVE 99 TO UPPER-I .
 I-CREDIT-ANALYSIS-05.
      CALL "INPUTI" USING ROW, COLUMN,
      LOWER-I, UPPER-I, INPUT-I.
      MOVE INPUT-I TO TP-CREDIT-ANALYSIS.
 I-CREDIT-ANALYSIS-99.
      EXIT.
*
 INPUT-CREDIT-NARR SECTION.
 I-CREDIT-NARR-00.
      MOVE 1 TO COLUMN.
 I-CREDIT-NARR-05.
      MOVE 50 TO ST-SIZE.
      CALL "INPUTS" USING ROW, COLUMN,
      INPUT-S, ST-SIZE.
      MOVE INPUT-S TO TP-CREDIT-NARR.
 I-CREDIT-NARR-99.
      EXIT.
*
 INPUT-DEBIT-AMOUNT SECTION.
 I-DEBIT-AMOUNT-00.
      MOVE 1 TO COLUMN.
      MOVE ZERO TO LOWER-R.
      MOVE 999999999.99 TO UPPER-R .
      MOVE 2 TO DECIMALS.
 I-DEBIT-AMOUNT-05.
      CALL "INPUTRT" USING ROW, COLUMN,
      LOWER-R, UPPER-R, INPUT-R, INPUT-S, DECIMALS.
      IF INPUT-S = SPACE
```

```
            MOVE "N" TO FLAG-RECORD
            GO TO I-DEBIT-AMOUNT-99.
        MOVE SPACE TO FLAG-RECORD.
        MOVE INPUT-R TO TP-DEBIT-AMOUNT.
 I-DEBIT-AMOUNT-99.
        EXIT.
*
 INPUT-DEBIT-ACCOUNT SECTION.
 I-DEBIT-ACCOUNT-00.
        MOVE 14 TO COLUMN.
 I-DEBIT-ACCOUNT-05.
        MOVE 7 TO ST-SIZE.
        CALL "INPUTS" USING ROW, COLUMN,
        INPUT-S, ST-SIZE.
        MOVE INPUT-S TO TP-DEBIT-ACCOUNT.
 I-DEBIT-ACCOUNT-99.
        EXIT.
*
 INPUT-DEBIT-ANALYSIS SECTION.
 I-DEBIT-ANALYSIS-00.
        MOVE 22 TO COLUMN.
        MOVE ZERO TO LOWER-I.
        MOVE 99 TO UPPER-I .
 I-DEBIT-ANALYSIS-05.
        CALL "INPUTI" USING ROW, COLUMN,
        LOWER-I, UPPER-I, INPUT-I.
        MOVE INPUT-I TO TP-DEBIT-ANALYSIS.
 I-DEBIT-ANALYSIS-99.
        EXIT.
*
 INPUT-DEBIT-NARR SECTION.
 I-DEBIT-NARR-00.
        MOVE 1 TO COLUMN.
 I-DEBIT-NARR-05.
        MOVE 50 TO ST-SIZE.
        CALL "INPUTS" USING ROW, COLUMN,
        INPUT-S, ST-SIZE.
        MOVE INPUT-S TO TP-DEBIT-NARR.
 I-DEBIT-NARR-99.
        EXIT.
*  the following sections allow the display of the
*    contents of individual fields on the TP work
*    file.
*
*
*
 DISPLAY-HEADER-YEAR SECTION.
 D-HEADER-YEAR-00.
        MOVE 1 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY TP-HEADER-YEAR WITH NO ADVANCING.
 D-HEADER-YEAR-99.
        EXIT.
*
 DISPLAY-HEADER-DATE SECTION.
 D-HEADER-DATE-00.
        MOVE 6 TO COLUMN.
        IF TP-HEADER-DATE NUMERIC
            MOVE TP-HEADER-DATE TO OUTPUT-I
```

```
            CALL "OUTPUTI" USING ROW, COLUMN, OUTPUT-I
        ELSE
            MOVE "Field not numeric" TO ERROR-MESSAGE
            GO TO FAIL-END.
 D-HEADER-DATE-99.
        EXIT.
*
 DISPLAY-HEADER-CREDIT SECTION.
 D-HEADER-CREDIT-00.
        MOVE 13 TO COLUMN.
        IF TP-HEADER-CREDIT NUMERIC
            MOVE TP-HEADER-CREDIT TO OUTPUT-R
            CALL "OUTPUTR" USING ROW, COLUMN, OUTPUT-R
        ELSE
            MOVE "Field not numeric" TO ERROR-MESSAGE
            GO TO FAIL-END.
 D-HEADER-CREDIT-99.
        EXIT.
*
 DISPLAY-HEADER-DEBIT SECTION.
 D-HEADER-DEBIT-00.
        MOVE 26 TO COLUMN.
        IF TP-HEADER-DEBIT NUMERIC
            MOVE TP-HEADER-DEBIT TO OUTPUT-R
            CALL "OUTPUTR" USING ROW, COLUMN, OUTPUT-R
        ELSE
            MOVE "Field not numeric" TO ERROR-MESSAGE
            GO TO FAIL-END.
 D-HEADER-DEBIT-99.
        EXIT.
*
 DISPLAY-HEADER-TYPE SECTION.
 D-HEADER-TYPE-00.
        MOVE 39 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY TP-HEADER-TYPE WITH NO ADVANCING.
 D-HEADER-TYPE-99.
        EXIT.
*
 DISPLAY-CREDIT-AMOUNT SECTION.
 D-CREDIT-AMOUNT-00.
        MOVE 1 TO COLUMN.
        IF TP-CREDIT-AMOUNT NUMERIC
            MOVE TP-CREDIT-AMOUNT TO OUTPUT-R
            CALL "OUTPUTR" USING ROW, COLUMN, OUTPUT-R
        ELSE
            MOVE "Field not numeric" TO ERROR-MESSAGE
            GO TO FAIL-END.
 D-CREDIT-AMOUNT-99.
        EXIT.
*
 DISPLAY-CREDIT-ACCOUNT SECTION.
 D-CREDIT-ACCOUNT-00.
        MOVE 14 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY TP-CREDIT-ACCOUNT WITH NO ADVANCING.
 D-CREDIT-ACCOUNT-99.
        EXIT.
*
```

```
 DISPLAY-CREDIT-ANALYSIS SECTION.
 D-CREDIT-ANALYSIS-00.
     MOVE 22 TO COLUMN.
     IF TP-CREDIT-ANALYSIS NUMERIC
         MOVE TP-CREDIT-ANALYSIS TO OUTPUT-I
         CALL "OUTPUTI" USING ROW, COLUMN, OUTPUT-I
     ELSE
         MOVE "Field not numeric" TO ERROR-MESSAGE
         GO TO FAIL-END.
 D-CREDIT-ANALYSIS-99.
     EXIT.
*
 DISPLAY-CREDIT-NARR SECTION.
 D-CREDIT-NARR-00.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY TP-CREDIT-NARR WITH NO ADVANCING.
 D-CREDIT-NARR-99.
     EXIT.
*
 DISPLAY-DEBIT-AMOUNT SECTION.
 D-DEBIT-AMOUNT-00.
     MOVE 1 TO COLUMN.
     IF TP-DEBIT-AMOUNT NUMERIC
         MOVE TP-DEBIT-AMOUNT TO OUTPUT-R
         CALL "OUTPUTR" USING ROW, COLUMN, OUTPUT-R
     ELSE
         MOVE "Field not numeric" TO ERROR-MESSAGE
         GO TO FAIL-END.
 D-DEBIT-AMOUNT-99.
     EXIT.
*
 DISPLAY-DEBIT-ACCOUNT SECTION.
 D-DEBIT-ACCOUNT-00.
     MOVE 14 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY TP-DEBIT-ACCOUNT WITH NO ADVANCING.
 D-DEBIT-ACCOUNT-99.
     EXIT.
*
 DISPLAY-DEBIT-ANALYSIS SECTION.
 D-DEBIT-ANALYSIS-00.
     MOVE 22 TO COLUMN.
     IF TP-DEBIT-ANALYSIS NUMERIC
         MOVE TP-DEBIT-ANALYSIS TO OUTPUT-I
         CALL "OUTPUTI" USING ROW, COLUMN, OUTPUT-I
     ELSE
         MOVE "Field not numeric" TO ERROR-MESSAGE
         GO TO FAIL-END.
 D-DEBIT-ANALYSIS-99.
     EXIT.
*
 DISPLAY-DEBIT-NARR SECTION.
 D-DEBIT-NARR-00.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY TP-DEBIT-NARR WITH NO ADVANCING.
 D-DEBIT-NARR-99.
     EXIT.
```

```
*
* this controls the displaying of the background
*   details on a selected screen page.
*
*
*
 NEW-HEADER-PAGE SECTION.
 NP-HEADER-00.
     MOVE 5 TO CURRENT-LINE.
     MOVE 6 TO SCROLL-FROM.
     MOVE 21 TO SCROLL-TO.
     MOVE 4 TO ROW.
     CALL "CLEARVDU" USING ROW, COLUMN.
     DISPLAY "year" WITH NO ADVANCING.
     MOVE 6 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "date" WITH NO ADVANCING.
     MOVE 13 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "credit" WITH NO ADVANCING.
     MOVE 26 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "debit" WITH NO ADVANCING.
     MOVE 39 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "type" WITH NO ADVANCING.
 NP-HEADER-99.
     EXIT.
*
 NEW-CREDIT-PAGE SECTION.
 NP-CREDIT-00.
     MOVE 6 TO CURRENT-LINE.
     MOVE 7 TO SCROLL-FROM.
     MOVE 21 TO SCROLL-TO.
     MOVE 4 TO ROW.
     CALL "CLEARVDU" USING ROW, COLUMN.
     DISPLAY "amount" WITH NO ADVANCING.
     MOVE 14 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "account" WITH NO ADVANCING.
     MOVE 22 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "analysis" WITH NO ADVANCING.
     MOVE 5 TO ROW.
     MOVE 1 TO COLUMN.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY "narrative" WITH NO ADVANCING.
 NP-CREDIT-99.
     EXIT.
*
 NEW-DEBIT-PAGE SECTION.
 NP-DEBIT-00.
     MOVE 6 TO CURRENT-LINE.
     MOVE 7 TO SCROLL-FROM.
     MOVE 21 TO SCROLL-TO.
     MOVE 4 TO ROW.
     CALL "CLEARVDU" USING ROW, COLUMN.
     DISPLAY "amount" WITH NO ADVANCING.
     MOVE 14 TO COLUMN.
```

```
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "account" WITH NO ADVANCING.
        MOVE 22 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "analysis" WITH NO ADVANCING.
        MOVE 5 TO ROW.
        MOVE 1 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "narrative" WITH NO ADVANCING.
 NP-DEBIT-99.
        EXIT.
*
* scroll the input area
*
 SCROLL-AREA SECTION.
 SCR00.
        CALL "SCROLL" USING SCROLL-FROM, SCROLL-TO.
        MOVE SCROLL-TO TO ROW.
        MOVE 80 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "  ".
        MOVE SCROLL-FROM TO ST-FROM.
        MOVE SCROLL-TO TO ST-TO.
        MOVE 1 TO SCROLL-FROM.
        MOVE 24 TO SCROLL-TO.
        CALL "SCROLL" USING SCROLL-FROM, SCROLL-TO.
        MOVE ST-FROM TO SCROLL-FROM.
        MOVE ST-TO TO SCROLL-TO.
 SCR99.
        EXIT.
*
*  this asks on the 23rd line whether the change is to be
*    applied to the file. It is asked (i) after amendment of
*    the fields in the insert function (ii) after amendment
*    of the fields in the amend function (iii) after viewing
*    all the contents of the record in the delete function.
*  It sets the variable REPLY.
*
 INQUIRE-OK SECTION.
 IQ00.
        MOVE 23 TO ROW.
        MOVE 1 TO COLUMN.
        CALL "CURSORTO" USING ROW, COLUMN.
        DISPLAY "Apply the change (Y=Yes)" WITH NO ADVANCING.
        MOVE 28 TO COLUMN.
        CALL "INPUTS" USING ROW, COLUMN, INPUT-S, ST-SIZE.
        CALL "CLEARVDU" USING ROW.
        IF IXX (1) = "Y" OR IXX (1) = "y"
            MOVE 4 TO REPLY
        ELSE
            MOVE ZERO TO REPLY.
 IQ99.
        EXIT.
*
* this forces the program to fail with an error being
*  returned to the operating system.
*
 FAIL-END SECTION.
 FA00.
```

```
    CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE,
        ERROR-MESSAGE, INPUT-S.
    MOVE "PROGRAM RUN ABANDONED" TO ERROR-MESSAGE
    CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE,
        ERROR-MESSAGE, INPUT-S.
    MOVE ZERO TO WORK-1.
    DIVIDE 2 BY WORK-1 GIVING WORK-2.
    PERFORM CLOSEDOWN.
    STOP "PROGRAM RUN ABANDONED".
FA99.
    EXIT.
```

Batch Data Output/Format Program

This program produced by the generator is not the final product. It processes and audits the data in the batches, checking for format and total errors but does not do any updating onto the system with the data. This coding is required to be added by the programmer. The relevant codings are inserted into the sections PROCESS-<record> SECTION.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TOJOURNAL.
*
* This program was produced by a tool written
*   as part of a Master of Science
*   postgraduate thesis by John N Sutherland
*   under the supervision of Professor A.J. Cole
*   at the Department of Computational Science,
*   The University of St. Andrews, Scotland.
*
* This program processes the tp file and prints
*   the contents.
*
* Screen layouts, print layouts, user operating
*   instructions are also produced along with
*   the program to handle the data entry
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
*
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
*
* the file TP-FILE is the data entry file
*
     SELECT TP-FILE ASSIGN TO TPFILE
         ORGANIZATION IS INDEXED
         ACCESS IS DYNAMIC
         RECORD KEY IS TP-KEY.
     SELECT AUDIT ASSIGN TO PRINTER.
 DATA DIVISION.
 FILE SECTION.
 FD TP-FILE.
 01 TP-KEY.
     03 TP-BATCH-NUMBER          PIC 9999.
     03 TP-RECORD-NUMBER         PIC 99.
     03 TP-RECORD-TYPE           PIC X(10).
     03 TP-LINE-NUMBER           PIC 9999.
 01  TP-HEADER-RECORD.
     03  FILLER                  PIC X(20).
```

```
        03  TP-HEADER-YEAR              PIC X.
        03  TP-HEADER-DATE              PIC 9(6).
        03  TP-HEADER-CREDIT            PIC 9(9)V9(2).
        03  TP-HEADER-DEBIT             PIC 9(9)V9(2).
        03  TP-HEADER-TYPE              PIC X.
    01  TP-CREDIT-RECORD.
        03  FILLER                      PIC X(20).
        03  TP-CREDIT-AMOUNT            PIC 9(9)V9(2).
        03  TP-CREDIT-ACCOUNT           PIC X(7).
        03  TP-CREDIT-ANALYSIS          PIC 9(2).
        03  TP-CREDIT-NARR              PIC X(50).
    01  TP-DEBIT-RECORD.
        03  FILLER                      PIC X(20).
        03  TP-DEBIT-AMOUNT             PIC 9(9)V9(2).
        03  TP-DEBIT-ACCOUNT            PIC X(7).
        03  TP-DEBIT-ANALYSIS           PIC 9(2).
        03  TP-DEBIT-NARR               PIC X(50).
    FD  AUDIT.
    01  AUDIT-RECORD.
        03  AUDIT-NAME                  PIC X(50).
        03  AUDIT-VALUE-X               PIC X(30).
        03  FILLER REDEFINES AUDIT-VALUE-X.
            05  AUDIT-VALUE-9           PIC -(17)9.
            05  FILLER                  PIC X(12).
        03  FILLER REDEFINES AUDIT-VALUE-X.
            05  AUDIT-VALUE-R           PIC -(12)9.9(5).
            05  FILLER                  PIC X(11).
        03  FILLER REDEFINES AUDIT-VALUE-X.
            05  AUDIT-DAY               PIC 99.
            05  FILLER                  PIC X.
            05  AUDIT-MONTH             PIC 99.
            05  FILLER                  PIC X.
            05  AUDIT-YEAR              PIC 99.
            05  FILLER                  PIC XXX.
            05  AUDIT-HOUR              PIC 99.
            05  FILLER                  PIC X.
            05  AUDIT-MINUTE            PIC 99.
            05  FILLER                  PIC X.
            05  AUDIT-SECOND            PIC 99.
            05  FILLER                  PIC XXX.
            05  AUDIT-PAGE-LITERAL      PIC XXXX.
            05  AUDIT-PAGE              PIC ZZ9.
    01  AUDIT-BATCH.
        03  AB-LITERAL                  PIC X(5).
        03  FILLER                      PIC XX.
        03  AB-BATCH                    PIC ZZZ9.
    01  FILLER.
        03  AH-HEADER-YEAR               PIC X(4).
        03  FILLER                       PIC X.
        03  AH-HEADER-DATE               PIC X(4).
        03  FILLER                       PIC X(3).
        03  AH-HEADER-CREDIT             PIC X(6).
        03  FILLER                       PIC X(7).
        03  AH-HEADER-DEBIT              PIC X(5).
        03  FILLER                       PIC X(8).
        03  AH-HEADER-TYPE               PIC X(4).
    01  FILLER.
        03  AR-HEADER-YEAR               PIC X.
        03  FILLER                       PIC X(4).
```

```
      03  AR-HEADER-DATE              PIC 9(6).
      03  FILLER                      PIC X.
      03  AR-HEADER-CREDIT            PIC Z(8)9.9(2).
      03  FILLER                      PIC X.
      03  AR-HEADER-DEBIT             PIC Z(8)9.9(2).
      03  FILLER                      PIC X.
      03  AR-HEADER-TYPE              PIC X.
  01  FILLER.
      03  AH-CREDIT-AMOUNT            PIC X(6).
      03  FILLER                      PIC X(7).
      03  AH-CREDIT-ACCOUNT           PIC X(7).
      03  FILLER                      PIC X.
      03  AH-CREDIT-ANALYSIS          PIC X(8).
  01  FILLER.
      03  AH-CREDIT-NARR              PIC X(9).
  01  FILLER.
      03  AR-CREDIT-AMOUNT            PIC Z(8)9.9(2).
      03  FILLER                      PIC X.
      03  AR-CREDIT-ACCOUNT           PIC X(7).
      03  FILLER                      PIC X.
      03  AR-CREDIT-ANALYSIS          PIC Z9.
  01  FILLER.
      03  AR-CREDIT-NARR              PIC X(50).
  01  FILLER.
      03  AH-DEBIT-AMOUNT             PIC X(6).
      03  FILLER                      PIC X(7).
      03  AH-DEBIT-ACCOUNT            PIC X(7).
      03  FILLER                      PIC X.
      03  AH-DEBIT-ANALYSIS           PIC X(8).
  01  FILLER.
      03  AH-DEBIT-NARR               PIC X(9).
  01  FILLER.
      03  AR-DEBIT-AMOUNT             PIC Z(8)9.9(2).
      03  FILLER                      PIC X.
      03  AR-DEBIT-ACCOUNT            PIC X(7).
      03  FILLER                      PIC X.
      03  AR-DEBIT-ANALYSIS           PIC Z9.
  01  FILLER.
      03  AR-DEBIT-NARR               PIC X(50).
WORKING-STORAGE SECTION.
  01  W-COUNT-HEADER-CREDIT          PIC 9(9)V9(2) COMP.
  01  W-STORE-HEADER-CREDIT          PIC 9(9)V9(2) COMP.
  01  W-COUNT-HEADER-DEBIT           PIC 9(9)V9(2) COMP.
  01  W-STORE-HEADER-DEBIT           PIC 9(9)V9(2) COMP.
  01  WORK-1                 PIC 9 COMP.
  01  WORK-2                 PIC 9 COMP.
  01  ST-BATCH-NUMBER        PIC 9999 VALUE ZERO.
  01  ST-AUDIT-RECORD        PIC X(80).
  01  LINE-COUNT             PIC 99 COMP VALUE 100.
  01  PAGE-COUNT             PIC 999 COMP VALUE 1.
  01  ST-DATE-TIME.
      03  ST-DATE            PIC 9(6).
      03  FILLER REDEFINES ST-DATE.
          05  ST-YEAR        PIC 99.
          05  ST-MONTH       PIC 99.
          05  ST-DAY         PIC 99.
      03  ST-TIME            PIC 9(8).
      03  FILLER REDEFINES ST-TIME.
          05  ST-HOUR        PIC 99.
```

```
          05  ST-MINUTE    PIC 99.
          05  ST-SECOND    PIC 99.
          05  FILLER       PIC XX.
 01   AUDIT-ADVANCE              PIC 9999 COMP.
 01   END-FLAG                PIC 9 COMP VALUE ZERO.
 01   ST-RECORD-TYPE            PIC X(10) VALUE SPACES.
*
 PROCEDURE DIVISION.
 CONTROLX SECTION.
 C00.
     PERFORM INITIALX.
     PERFORM MAINFLOW.
     PERFORM CLOSEDOWN.
 C99.
     STOP RUN.
*
 INITIALX SECTION.
*
* open the files; store the program title
*
 I00.
     OPEN INPUT TP-FILE.
     OPEN OUTPUT AUDIT.
     MOVE SPACES TO AUDIT-RECORD.
 I99.
     EXIT.
*
*
 MAINFLOW SECTION.
 M00.
     READ TP-FILE NEXT AT END
          MOVE "NO RECORDS ON FILE" TO AUDIT-RECORD
          MOVE 1 TO AUDIT-ADVANCE
          PERFORM AUDIT-PRINT
          GO TO M99.
 M05.
     IF TP-BATCH-NUMBER NOT = ZERO
     AND TP-BATCH-NUMBER NOT = ST-BATCH-NUMBER
          MOVE TP-BATCH-NUMBER TO ST-BATCH-NUMBER
          MOVE "Batch" TO AB-LITERAL
          MOVE TP-BATCH-NUMBER TO AB-BATCH
          MOVE 2 TO AUDIT-ADVANCE
          PERFORM AUDIT-PRINT
          MOVE 2 TO AUDIT-ADVANCE.
     READ TP-FILE NEXT AT END
          MOVE "EMPTY LAST BATCH" TO AUDIT-RECORD
          MOVE 1 TO AUDIT-ADVANCE
          PERFORM AUDIT-PRINT
          GO TO FAIL-END.
 M10.
     IF TP-RECORD-TYPE NOT = ST-RECORD-TYPE
          IF TP-RECORD-TYPE = "HEADER    "
              PERFORM TYPE-HEADER
          ELSE
          IF TP-RECORD-TYPE = "CREDIT    "
              PERFORM TYPE-CREDIT
          ELSE
          IF TP-RECORD-TYPE = "DEBIT     "
              PERFORM TYPE-DEBIT
```

```
            ELSE
                MOVE "INVALID RECORD TYPE" TO AUDIT-RECORD
                MOVE 3 TO AUDIT-ADVANCE
                PERFORM AUDIT-PRINT
                MOVE TP-KEY TO AUDIT-RECORD
                PERFORM AUDIT-PRINT
                GO TO FAIL-END.
        MOVE TP-RECORD-TYPE TO ST-RECORD-TYPE.
        IF TP-RECORD-TYPE = "HEADER     "
            PERFORM PROCESS-HEADER
        ELSE
        IF TP-RECORD-TYPE = "CREDIT     "
            PERFORM PROCESS-CREDIT
        ELSE
        IF TP-RECORD-TYPE = "DEBIT      "
            PERFORM PROCESS-DEBIT
        ELSE
            MOVE "INVALID RECORD TYPE" TO AUDIT-RECORD
            MOVE 3 TO AUDIT-ADVANCE
            PERFORM AUDIT-PRINT
            MOVE TP-KEY TO AUDIT-RECORD
            PERFORM AUDIT-PRINT
            GO TO FAIL-END.
        READ TP-FILE NEXT AT END
            MOVE 1 TO END-FLAG
            GO TO M15.
        IF ST-BATCH-NUMBER = TP-BATCH-NUMBER
            GO TO M10.
    M15.
        IF W-STORE-HEADER-CREDIT NOT = W-COUNT-HEADER-CREDIT
            MOVE "CONTROL FAILURE" TO AUDIT-VALUE-X
            MOVE 5 TO AUDIT-ADVANCE
            PERFORM AUDIT-PRINT
            MOVE "credit count incorrect" TO AUDIT-VALUE-X
            PERFORM AUDIT-PRINT
            MOVE W-STORE-HEADER-CREDIT TO AUDIT-VALUE-R
            PERFORM AUDIT-PRINT
            MOVE W-COUNT-HEADER-CREDIT TO AUDIT-VALUE-R
            PERFORM AUDIT-PRINT
            GO TO FAIL-END.
        IF W-STORE-HEADER-DEBIT NOT = W-COUNT-HEADER-DEBIT
            MOVE "CONTROL FAILURE" TO AUDIT-VALUE-X
            MOVE 5 TO AUDIT-ADVANCE
            PERFORM AUDIT-PRINT
            MOVE "debit count incorrect" TO AUDIT-VALUE-X
            PERFORM AUDIT-PRINT
            MOVE W-STORE-HEADER-DEBIT TO AUDIT-VALUE-R
            PERFORM AUDIT-PRINT
            MOVE W-COUNT-HEADER-DEBIT TO AUDIT-VALUE-R
            PERFORM AUDIT-PRINT
            GO TO FAIL-END.
        IF END-FLAG = ZERO
            INSPECT AUDIT-RECORD REPLACING ALL SPACES BY "-"
            MOVE 2 TO AUDIT-ADVANCE
            PERFORM AUDIT-PRINT
            GO TO M05.
    M99.
        EXIT.
    *
```

```
*  close files before the end of the
*  run
*
 CLOSEDOWN SECTION.
 C00.
      MOVE "End of Input Report" TO AUDIT-RECORD.
      MOVE 2 TO AUDIT-ADVANCE.
      PERFORM AUDIT-PRINT.
      CLOSE AUDIT.
      CLOSE TP-FILE.
 C99.
      EXIT.
*
*  these control processing of record types
*
*
  PROCESS-HEADER SECTION.
  PR-HEADER-00.
      PERFORM AUDIT-HEADER.
      MOVE TP-HEADER-CREDIT TO W-STORE-HEADER-CREDIT.
      MOVE ZERO TO W-COUNT-HEADER-CREDIT.
      MOVE TP-HEADER-DEBIT TO W-STORE-HEADER-DEBIT.
      MOVE ZERO TO W-COUNT-HEADER-DEBIT.
  PR-HEADER-99.
      EXIT.
 *
  PROCESS-CREDIT SECTION.
  PR-CREDIT-00.
      PERFORM AUDIT-CREDIT.
      ADD TP-CREDIT-AMOUNT TO W-COUNT-HEADER-CREDIT.
  PR-CREDIT-99.
      EXIT.
 *
  PROCESS-DEBIT SECTION.
  PR-DEBIT-00.
      PERFORM AUDIT-DEBIT.
      ADD TP-DEBIT-AMOUNT TO W-COUNT-HEADER-DEBIT.
  PR-DEBIT-99.
      EXIT.
*
*  this forces the program to fail with an error being
*   returned to the operating system.
*
 FAIL-END SECTION.
 FA00.
      MOVE "PROGRAM RUN ABANDONED" TO AUDIT-RECORD.
      MOVE 2 TO AUDIT-ADVANCE.
      PERFORM AUDIT-PRINT.
      MOVE ZERO TO WORK-1.
      DIVIDE 2 BY WORK-1 GIVING WORK-2.
      PERFORM CLOSEDOWN.
      STOP "PROGRAM RUN ABANDONED".
 FA99.
      EXIT.
*
*  common routine for writing a record to the audit file
*
 AUDIT-PRINT SECTION.
 AU00.
```

```
     IF LINE-COUNT > 55
         PERFORM AUDIT-HEADINGS.
     WRITE AUDIT-RECORD AFTER ADVANCING AUDIT-ADVANCE LINES.
     MOVE SPACES TO AUDIT-RECORD.
     ADD AUDIT-ADVANCE TO LINE-COUNT.
     MOVE 1 TO AUDIT-ADVANCE.
 AU99.
      EXIT.
*
* head the audit print and reset the line and page
*   counters
*
 AUDIT-HEADINGS SECTION.
 AH00.
     MOVE AUDIT-RECORD TO ST-AUDIT-RECORD.
     MOVE SPACES TO AUDIT-RECORD.
     MOVE "Journal, Salary, Wage"
         TO AUDIT-NAME.
     MOVE "  /  /      ¦ :  :      Page" TO AUDIT-VALUE-X.
     MOVE PAGE-COUNT TO AUDIT-PAGE.
     ACCEPT ST-DATE FROM DATE.
     ACCEPT ST-TIME FROM TIME.
     MOVE ST-YEAR TO AUDIT-YEAR.
     MOVE ST-MONTH TO AUDIT-MONTH.
     MOVE ST-DAY TO AUDIT-DAY.
     MOVE ST-HOUR TO AUDIT-HOUR.
     MOVE ST-MINUTE TO AUDIT-MINUTE.
     MOVE ST-SECOND TO AUDIT-SECOND.
     WRITE AUDIT-RECORD AFTER ADVANCING PAGE.
     MOVE SPACES TO AUDIT-RECORD.
     WRITE AUDIT-RECORD AFTER ADVANCING 2 LINES.
     MOVE 1 TO LINE-COUNT.
     ADD 1 TO PAGE-COUNT.
     MOVE ST-AUDIT-RECORD TO AUDIT-RECORD.
     MOVE SPACES TO ST-RECORD-TYPE.
 AH99.
     EXIT.
*
* these print the contents of the record fields
*
*
 AUDIT-HEADER SECTION.
 AUD-HEADER-00.
     MOVE TP-HEADER-YEAR TO AR-HEADER-YEAR.
     IF TP-HEADER-DATE NUMERIC
         MOVE TP-HEADER-DATE TO AR-HEADER-DATE
     ELSE
         GO TO AUD-HEADER-90.
     IF TP-HEADER-CREDIT NUMERIC
         MOVE TP-HEADER-CREDIT TO AR-HEADER-CREDIT
     ELSE
         GO TO AUD-HEADER-90.
     IF TP-HEADER-DEBIT NUMERIC
         MOVE TP-HEADER-DEBIT TO AR-HEADER-DEBIT
     ELSE
         GO TO AUD-HEADER-90.
     MOVE TP-HEADER-TYPE TO AR-HEADER-TYPE.
     PERFORM AUDIT-PRINT.
     GO TO AUD-HEADER-99.
```

```
 AUD-HEADER-90.
     MOVE "Invalid data" TO AUDIT-NAME.
     PERFORM AUDIT-PRINT.
     GO TO FAIL-END.
 AUD-HEADER-99.
     EXIT.
*
 AUDIT-CREDIT SECTION.
 AUD-CREDIT-00.
     IF TP-CREDIT-AMOUNT NUMERIC
         MOVE TP-CREDIT-AMOUNT TO AR-CREDIT-AMOUNT
     ELSE
         GO TO AUD-CREDIT-90.
     MOVE TP-CREDIT-ACCOUNT TO AR-CREDIT-ACCOUNT.
     IF TP-CREDIT-ANALYSIS NUMERIC
         MOVE TP-CREDIT-ANALYSIS TO AR-CREDIT-ANALYSIS
     ELSE
         GO TO AUD-CREDIT-90.
     PERFORM AUDIT-PRINT.
     MOVE TP-CREDIT-NARR TO AR-CREDIT-NARR.
     PERFORM AUDIT-PRINT.
     GO TO AUD-CREDIT-99.
 AUD-CREDIT-90.
     MOVE "Invalid data" TO AUDIT-NAME.
     PERFORM AUDIT-PRINT.
     GO TO FAIL-END.
 AUD-CREDIT-99.
     EXIT.
*
 AUDIT-DEBIT SECTION.
 AUD-DEBIT-00.
     IF TP-DEBIT-AMOUNT NUMERIC
         MOVE TP-DEBIT-AMOUNT TO AR-DEBIT-AMOUNT
     ELSE
         GO TO AUD-DEBIT-90.
     MOVE TP-DEBIT-ACCOUNT TO AR-DEBIT-ACCOUNT.
     IF TP-DEBIT-ANALYSIS NUMERIC
         MOVE TP-DEBIT-ANALYSIS TO AR-DEBIT-ANALYSIS
     ELSE
         GO TO AUD-DEBIT-90.
     PERFORM AUDIT-PRINT.
     MOVE TP-DEBIT-NARR TO AR-DEBIT-NARR.
     PERFORM AUDIT-PRINT.
     GO TO AUD-DEBIT-99.
 AUD-DEBIT-90.
     MOVE "Invalid data" TO AUDIT-NAME.
     PERFORM AUDIT-PRINT.
     GO TO FAIL-END.
 AUD-DEBIT-99.
     EXIT.
*
 TYPE-HEADER SECTION.
 T-HEADER-00.
     MOVE 2 TO AUDIT-ADVANCE.
     MOVE "HEADER:" TO AUDIT-RECORD.
     PERFORM AUDIT-PRINT.
     MOVE "year" TO AH-HEADER-YEAR.
     MOVE "date" TO AH-HEADER-DATE.
     MOVE "credit" TO AH-HEADER-CREDIT.
```

```
     MOVE "debit" TO AH-HEADER-DEBIT.
     MOVE "type" TO AH-HEADER-TYPE.
     PERFORM AUDIT-PRINT.
     MOVE 2 TO AUDIT-ADVANCE.
 T-HEADER-99.
     EXIT.
*
 TYPE-CREDIT SECTION.
 T-CREDIT-00.
     MOVE 2 TO AUDIT-ADVANCE.
     MOVE "CREDIT:" TO AUDIT-RECORD.
     PERFORM AUDIT-PRINT.
     MOVE "amount" TO AH-CREDIT-AMOUNT.
     MOVE "account" TO AH-CREDIT-ACCOUNT.
     MOVE "analysis" TO AH-CREDIT-ANALYSIS.
     PERFORM AUDIT-PRINT.
     MOVE "narrative" TO AH-CREDIT-NARR.
     PERFORM AUDIT-PRINT.
     MOVE 2 TO AUDIT-ADVANCE.
 T-CREDIT-99.
     EXIT.
*
 TYPE-DEBIT SECTION.
 T-DEBIT-00.
     MOVE 2 TO AUDIT-ADVANCE.
     MOVE "DEBIT:" TO AUDIT-RECORD.
     PERFORM AUDIT-PRINT.
     MOVE "amount" TO AH-DEBIT-AMOUNT.
     MOVE "account" TO AH-DEBIT-ACCOUNT.
     MOVE "analysis" TO AH-DEBIT-ANALYSIS.
     PERFORM AUDIT-PRINT.
     MOVE "narrative" TO AH-DEBIT-NARR.
     PERFORM AUDIT-PRINT.
     MOVE 2 TO AUDIT-ADVANCE.
 T-DEBIT-99.
     EXIT.
```

Documentation

The documentation is produced from the skeleton and parameters. It is produced in a format for input to the Digital Standard Runoff (DSR) text processing package. Below are two listings. The first is the documentation as produced by the documentation generator, the second the formatted documentation as produced by DSR.

Note (i) the print, vdu and operating documentation are in one file and can be subsequently separated if required (ii) the static lines are indicated by a vertical bar as before (only indicated on the unformatted documentation below).

The use of DSR format allows (i) the generator to be concerned only with outputting the correct text without concerning itself with the equally difficult task of formatting (ii) the documentation to be amended if required without loss of formatting.

Unformatted documentation

```
.nnm.page size 58,66
.lt
Program Documentation
=====================

     program: TOJOURNAL.COB          Input Print Layout

================================================================================

.el.pg.lt

Journal, Salary, Wage                        DD/MM/YY  HH:MM:SS   Page Z9

Batch  ZZZ9

HEADER:
year date   credit        debit         type

X    999999 999999999.99 999999999.99 X
<exactly 1 of this record>

CREDIT:
amount        account analysis
narrative

999999999.99 XXXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>

DEBIT:
amount        account analysis
narrative

999999999.99 XXXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>

--------------------------------------------------------------------------------

Batch  ZZZ9

<etc.>

End of Input Report
.el.pg.lt

Program Documentation
=====================

     program: TIJOURNAL.COB          Screen Layouts
```

```
.el.pg.lt


Journal, Salary, Wage Transaction Processing
Batch number 9999   <1> batch            <1> header
---------------------------------------------------------------------
year date   credit        debit         type

X    999999 999999999.99 999999999.99 X
<exactly 1 of this record>













<line 23 - prompts>
<line 24 - errors>

<1> = one of 'Insert' or 'Amend'

.el.pg.lt


Journal, Salary, Wage Transaction Processing
Batch number 9999   <1> batch            <1> credit
---------------------------------------------------------------------
amount        account analysis
narrative

999999999.99 XXXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>













<line 23 - prompts>
<line 24 - errors>
```

```
<1> = one of 'Insert' or 'Amend'

.el.pg.lt


Journal, Salary, Wage Transaction Processing
Batch number 9999   <1> batch           <1> debit
-------------------------------------------------------------------------
amount       account analysis
narrative

999999999.99 XXXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>









<line 23 - prompts>
<line 24 - errors>

<1> = one of 'Insert' or 'Amend'

¦.el.sp 2.lm 4.nmpg 1.FT
 .T  Journal, Salary and Wage Transaction Processing
¦.st Introduction
¦.pg.c;^&Introduction\&
¦.p;This suite allows you to add, amend, view or delete batches of
Journal, Salary and Wage information
¦for later processing.
¦The suite checks
¦the credit and debit totals automatically.
¦Another part of the suite
¦produces an audit of all data entered (but not all amendment and
¦deletions of batches).
¦.b2;^&The Screen\&
¦.p;The top line of the screen always displays the suite title.
¦.p;The second line tells the batch status: current batch number,
¦whether the batch is being entered for the first time or being amended,
¦what the current record type is and whether the current record is being
¦entered or amended.
¦.p;The next line just delimits these details from the details
¦being entered.
¦.p;Next are the heading lines that define the fields being entered.
¦.p;From here to the base of the screen is the area where data is
¦entered.
¦For details of the fields see 'Fields and their values' below. If any
¦entry involves more than one screen of data then this area of the screen
¦scrolls so that after you reach the last line, all subsequent entry is
¦from this line.
```

```
¦.p;The second last line is used to notify any messages or errors.
¦.p;The last line is used to ask any questions and receive replies.
¦.st Starting the run
¦.pg.c;^&Starting the run\&
¦.p;Each batch number must be unique. This determines whether you are
¦amending or inserting a batch. If a batch with the number you enter
¦exists then you are amending, otherwise entering.
¦.p;The program prompts for the batch number. Enter 'return' to end the
¦run. Otherwise enter the batch number required.
¦.p;If the batch exists then 'Amend batch' is displayed on the second
¦line. Follow the procedures for amending a batch as detailed below.
¦.p;If the batch does not exist then 'Insert batch' is displayed on the
¦second line. Follow the procedures for inserting a batch as detailed
¦in below.
¦.st Inserting a batch
¦.pg.c;^&Inserting a batch\&
 .pg.c;^&Header record\&
 .p;'Insert header' is displayed on the second line.
 .p;Enter the details of year, date, credit, debit and type (see 'Fields
and their values' below for details of field contents).
 .p;'Which field' is prompted on the last line of the screen. To change
a field's value enter its name as per the screen heading, e.g. enter
'year' to amend the year. Once or if all details are correct press
'return' to this prompt.
 .p;There is one record of this type.
 .pg.c;^&Credit record\&
 .p;'Insert credit' is displayed on the second line.
 .p;Enter the details of account, amount, analysis and narrative (see
'Fields and their values' below for details of field contents).
 .p;'Which field' is prompted on the last line of the screen. To change
a field's value enter its name as per the screen heading, e.g. enter
'account' to amend the account. Once or if all details are correct press
'return' to this prompt.
 .p;You may now enter another record.
 .p;To end entry of records enter 'return' only to the 'account' prompt
at either initial entry or on amending.
 .p;There are up to 9999 of these records.
 .pg.c;^&Debit record\&
 .p;'Insert debit' is displayed on the second line.
 .p;Enter the details of account, amount, analysis and narrative (see
'Fields and their values' below for details of field contents).
 .p;'Which field' is prompted on the last line of the screen. To change
a field's value enter its name as per the screen heading, e.g. enter
'account' to amend the account. Once or if all details are correct
press 'return' to this prompt.
 .p;You may now enter another record.
 .p;To end entry of records enter 'return' only to the 'account' prompt
at either initial entry or on amending.
 .p;There are up to 9999 of these records.
 .pg.c;^&Check totals\&
 .p;The program now checks the totals are correct.
 .p;If they are correct the program displays 'Batch totals correct' on
the second last line. Press return to acknowledge this. The program
returns to selecting another batch number (see above).
 .p;If the totals are not correct then the second last line displays
details of the total name, entered amount and accumulated amount. Press
return to acknowledge this error. You must now amend the batch as per
below.
¦.st Amend/Delete an existing batch
```

```
¦.pg.c;^&Amend/Delete an existing batch\&
¦.pg.p;'Amend or Delete this batch (D=Delete)' is displayed on the last
¦line.
¦.p;To delete the batch enter capital 'D'. There is one last check
¦before the batch is deleted. The program prompts 'Apply the Change
¦(Y=Yes)'. Reply 'Y' or 'y' to delete the batch. 'Batch deleted' is
¦displayed. Press 'return' to acknowledge this. The program returns to
¦select another batch number.
¦.p;To amend (or view) the batch press anything else.
 .pg.c;^&Header record\&
 .p;'Amend header' is displayed on the second line.
 .p;The details of year, date, debit, credit and type are displayed.
 .p;'Which field' is prompted on the last line of the screen.
 .p;If you do not wish to amend this record then enter 'return' here.
 .p;To change a field's value enter its name as per the screen heading,
e.g. enter 'year'to amend the year. Once all details are correct press
'return' to this prompt. The program prompts 'Apply the Change (Y=Yes)'.
Enter 'Y' or 'y' to apply the changes made; anything else to retain
the record as before the changes.
 .pg.c;^&Credit record\&
 .p;'Amend credit' is displayed on the second line.
 .p;^&Existing records\&
 .p;The details of amount, account, analysis and narrative are displayed.
 .p;'Which field' is prompted on the last line of the screen.
 .p;If you do not wish to amend this record then enter 'return' here.
If this is done then the program prompts 'Retain or Delete this
record'. Enter capital 'D' to delete the record; it is removed from
the screen. Enter anything else to continue with the next record.
 .p;To change a field's value enter its name as per the screen heading,
e.g. enter 'amount' amend the amount. Once all details are correct
press 'return' to this prompt. The program prompts 'Apply the Change
(Y=Yes)'. Enter 'Y' or 'y' to apply the changes made; anything else to
retain the record as before the changes.
 .p;If there are more records then their details are displayed for
possible amending.
 .p;^&Additional records\&
 .p;Once all existing records of this type have been viewed, amended or
deleted then the program prompts for entry of more records of this
type, if there are under 9999 of this type already.
 .p;The program prompts 'Add more records (N=No)'. To end entry of this
record type enter capital 'N'.
 .p;Any other entry to this prompt positions the cursor for entry of a
new record. Enter the details of amount, account, analysis and
narrative.
 .p;To the 'Which field' prompt the name of any field to be amended can
be entered as before e.g. enter 'amount' to amend  the amount.
 .p;The program then reprompts for another record.
 .p;To end the addition of new records either enter 'return' to
'account' at initial entry or on subsequent amending.
 .p;The program asks again 'Add more records (N=No)' if there are less
than 9999 of this type now on the batch.
 .pg.c;^&Debit record\&
 .p;'Amend debit' is displayed on the second line.
 .p;^&Existing records\&
 .p;The details of amount, account, analysis and narrative are displayed.
 .p;'Which field' is prompted on the last line of the screen.
 .p;If you do not wish to amend this record then enter 'return' here.
If this is done then the program prompts 'Retain or Delete this
record'. Enter capital 'D' to delete the record; it is removed from
```

the screen. Enter anything else to continue with the next record.
.p;To change a field's value enter its name as per the screen heading, e.g. enter 'amount' amend the amount. Once all details are correct press 'return' to this prompt. The program prompts 'Apply the Change (Y=Yes)'. Enter 'Y' or 'y' to apply the changes made; anything else to retain the record as before the changes.
.p;If there are more records then their details are displayed for possible amending.
.p;^&Additional records\&
.p;Once all existing records of this type have been viewed, amended or deleted then the program prompts for entry of more records of this type, if there are under 9999 of this type already.
.p;The program prompts 'Add more records (N=No)'. To end entry of this record type enter capital 'N'.
.p;Any other entry to this prompt positions the cursor for entry of a new record. Enter the details of amount, account, analysis and narrative.
.p;To the 'Which field' prompt the name of any field to be amended can be entered as before e.g. enter 'amount' to amend the amount.
.p;The program then reprompts for another record.
.p;To end the addition of new records either enter 'return' to 'account' at initial entry or on subsequent amending.
.p;The program asks again 'Add more records (N=No)' if there are less than 9999 of this type now on the batch.
.pg.c;^&Check totals\&
.p;The program now checks the totals are correct.
.p;If they are correct the program displays 'Batch totals correct' on the second last line. Press return to acknowledge this. The program returns to selecting another batch number (see above).
.p;If the totals are not correct then the second last line displays details of the total name, entered amount and accumulated amount. Press return to acknowledge this error. You must now amend the batch as per above.
.st Fields and their values
.pg.c;^&Fields and their Values\&
.pg.c;^&Header record\&
.b2;^&year\& : up to 2 numerics, ranging from ZERO to 99.
.b2;^&date\& : 6 numerics in DDMMYY format, ranging from 010150 to 311249.
.b2;^&credit\& : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.
.b2;^&debit\& : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.
.pg.c;^&Credit record\&
.b2;^&amount\& : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.
.b2;^&account\& : up to 7 characters.
.b2;^&analysis\& : up to two numerics, ranging from ZERO to 99.
.b2;^&narrative\& : up to 50 characters.
.pg.c;^&Debit record\&
.b2;^&amount\& : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.
.b2;^&account\& : up to 7 characters.
.b2;^&analysis\& : up to two numerics, ranging from ZERO to 99.
.b2;^&narrative\& : up to 50 characters.

Formatted documentation

Program Documentation

program: TOJOURNAL.COB Input Print Layout

```
Journal, Salary, Wage                    DD/MM/YY  HH:MM:SS   Page Z9

Batch  ZZZ9

HEADER:
year date   credit        debit         type

X    999999 999999999.99 999999999.99 X
<exactly 1 of this record>

CREDIT:
amount        account analysis
narrative

999999999.99 XXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>

DEBIT:
amount        account analysis
narrative

999999999.99 XXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>

-------------------------------------------------------------------------


Batch  ZZZ9

<etc.>

End of Input Report
```

Program Documentation

program: TIJOURNAL.COB Screen Layouts

```
Journal, Salary, Wage Transaction Processing
Batch number 9999   <1> batch          <1> header
-------------------------------------------------------------------------
year date   credit       debit        type

X    999999 999999999.99 999999999.99 X
<exactly 1 of this record>




<line 23 - prompts>
<line 24 - errors>

<1> = one of 'Insert' or 'Amend'
```

```
Journal, Salary, Wage Transaction Processing
Batch number 9999   <l> batch            <l> credit
-----------------------------------------------------------------
amount       account analysis
narrative

999999999.99 XXXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>








<line 23 - prompts>
<line 24 - errors>

<l> = one of 'Insert' or 'Amend'
```

```
Journal, Salary, Wage Transaction Processing
Batch number 9999   <1> batch            <1> debit
-------------------------------------------------------------------------
amount       account analysis
narrative

999999999.99 XXXXXXX 99
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<up to 9999 of these records>




<line 23 - prompts>
<line 24 - errors>

<1> = one of 'Insert' or 'Amend'
```

Introduction

This suite allows you to add, amend, view or delete batches of Journal, Salary and Wage information for later processing. The suite checks the credit and debit totals automatically. Another part of the suite produces an audit of all data entered (but not all amendment and deletions of batches).

The Screen

The top line of the screen always displays the suite title.

The second line tells the batch status: current batch number, whether the batch is being entered for the first time or being amended, what the current record type is and whether the current record is being entered or amended.

The next line just delimits these details from the details being entered.

Next are the heading lines that define the fields being entered.

From here to the base of the screen is the area where data is entered. For details of the fields see 'Fields and their values' below. If any entry involves more than one screen of data then this area of the screen scrolls so that after you reach the last line, all subsequent entry is from this line.

The second last line is used to notify any messages or errors.

The last line is used to ask any questions and receive replies.

Starting the run

Each batch number must be unique. This determines whether you are amending or inserting a batch. If a batch with the number you enter exists then you are amending, otherwise entering.

The program prompts for the batch number. Enter 'return' to end the run. Otherwise enter the batch number required.

If the batch exists then 'Amend batch' is displayed on the second line. Follow the procedures for amending a batch as detailed below.

If the batch does not exist then 'Insert batch' is displayed on the second line. Follow the procedures for inserting a batch as detailed in below.

Inserting a batch

Header record

'Insert header' is displayed on the second line.

Enter the details of year, date, credit, debit and type (see 'Fields and their values' below for details of field contents).

'Which field' is prompted on the last line of the screen. To change a field's value enter its name as per the screen heading, e.g. enter 'year' to amend the year. Once or if all details are correct press 'return' to this prompt.

There is one record of this type.

Credit record

'Insert credit' is displayed on the second line.

Enter the details of account, amount, analysis and narrative (see 'Fields and their values' below for details of field contents).

'Which field' is prompted on the last line of the screen. To change a field's value enter its name as per the screen heading, e.g. enter 'account' to amend the account. Once or if all details are correct press 'return' to this prompt.

You may now enter another record.

To end entry of records enter 'return' only to the 'account' prompt at either initial entry or on amending.

There are up to 9999 of these records.

Debit record

'Insert debit' is displayed on the second line.

Enter the details of account, amount, analysis and narrative (see 'Fields and their values' below for details of field contents).

'Which field' is prompted on the last line of the screen. To change a field's value enter its name as per the screen heading, e.g. enter 'account' to amend the account. Once or if all details are correct press 'return' to this prompt.

You may now enter another record.

To end entry of records enter 'return' only to the 'account' prompt at either initial entry or on amending.

There are up to 9999 of these records.

Check totals

The program now checks the totals are correct.

If they are correct the program displays 'Batch totals correct' on the second last line. Press return to acknowledge this. The program returns to selecting another batch number (see above).

If the totals are not correct then the second last line displays details of the total name, entered amount and accumulated amount. Press return to acknowledge this error. You must now amend the batch as per below.

Amend/Delete an existing batch

'Amend or Delete this batch (D=Delete)' is displayed on the last line.

To delete the batch enter capital 'D'. There is one last check before the batch is deleted. The program prompts 'Apply the Change (Y=Yes)'. Reply 'Y' or 'y' to delete the batch. 'Batch deleted' is displayed. Press 'return' to acknowledge this. The program returns to select another batch number.

To amend (or view) the batch press anything else.

Header record

'Amend header' is displayed on the second line.

The details of year, date, debit, credit and type are displayed.

'Which field' is prompted on the last line of the screen.

If you do not wish to amend this record then enter 'return' here.

To change a field's value enter its name as per the screen heading, e.g. enter 'year'to amend the year. Once all details are correct press 'return' to this prompt. The program prompts 'Apply the Change (Y=Yes)'. Enter 'Y' or 'y' to apply the changes made; anything else to retain the record as before the changes.

Credit record

'Amend credit' is displayed on the second line.

Existing records

The details of amount, account, analysis and narrative are displayed.

'Which field' is prompted on the last line of the screen.

If you do not wish to amend this record then enter 'return' here. If this is done then the program prompts 'Retain or Delete this record'. Enter capital 'D' to delete the record; it is removed from the screen. Enter anything else to continue with the next record.

To change a field's value enter its name as per the screen heading, e.g. enter 'amount' amend the amount. Once all details are correct press 'return' to this prompt. The program prompts 'Apply the Change (Y=Yes)'. Enter 'Y' or 'y' to apply the changes made; anything else to retain the record as before the changes.

If there are more records then their details are displayed for possible amending.

Additional records

Once all existing records of this type have been viewed, amended or deleted then the program prompts for entry of more records of this type, if there are under 9999 of this type already.

The program prompts 'Add more records (N=No)'. To end entry of this record type enter capital 'N'.

Any other entry to this prompt positions the cursor for entry of a new record. Enter the details of amount, account, analysis and narrative.

To the 'Which field' prompt the name of any field to be amended can be entered as before e.g. enter 'amount' to amend the amount.

The program then reprompts for another record.

To end the addition of new records either enter 'return' to 'account' at initial entry or on subsequent amending.

The program asks again 'Add more records (N=No)' if there are less than 9999 of this type now on the batch.

Debit record

'Amend debit' is displayed on the second line.

Existing records

The details of amount, account, analysis and narrative are displayed.

'Which field' is prompted on the last line of the screen.

If you do not wish to amend this record then enter 'return' here. If this is done then the program prompts 'Retain or Delete this record'. Enter capital 'D' to delete the record; it is removed from the screen. Enter anything else to continue with the next record.

To change a field's value enter its name as per the screen heading, e.g. enter 'amount' amend the amount. Once all details are correct press 'return' to this prompt. The program prompts 'Apply the Change (Y=Yes)'. Enter 'Y' or 'y' to apply the changes made; anything else to retain the record as before the changes.

If there are more records then their details are displayed for possible amending.

Additional records

Once all existing records of this type have been viewed, amended or deleted then the program prompts for entry of more records of this type, if there are under 9999 of this type already.

The program prompts 'Add more records (N=No)'. To end entry of this record type enter capital 'N'.

Any other entry to this prompt positions the cursor for entry of a new record. Enter the details of amount, account, analysis and narrative.

To the 'Which field' prompt the name of any field to be amended can be entered as before e.g. enter 'amount' to amend the amount.

The program then reprompts for another record.

To end the addition of new records either enter 'return' to 'account' at initial entry or on subsequent amending.

The program asks again 'Add more records (N=No)' if there are less than 9999 of this type now on the batch.

Check totals

The program now checks the totals are correct.

If they are correct the program displays 'Batch totals correct' on the second last line. Press return to acknowledge this. The program returns to selecting another batch number (see above).

If the totals are not correct then the second last line displays details of the total name, entered amount and accumulated amount. Press return to acknowledge this error. You must now amend the batch as per above.

Fields and their Values

Header record

year : up to 2 numerics, ranging from ZERO to 99.

date : 6 numerics in DDMMYY format, ranging from 010150 to 311249.

credit : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.

debit : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.

Credit record

amount : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.

account : up to 7 characters.

analysis : up to two numerics, ranging from ZERO to 99.

narrative : up to 50 characters.

Debit record

amount : up to 7 numerics before the decimal point 2 numerics after the decimal point, ranging from -9999999.99 to 9999999.99. This field is total checked.

account : up to 7 characters.

analysis : up to two numerics, ranging from ZERO to 99.

narrative : up to 50 characters.

4. Some of the COBOL Subroutines

As before the full details are held in an unbound appendix and tape available from the department. In general they emulate the facilities already added to s-algol.

There are 23 subroutines in total arranged in a hierarchy to avoid duplication as before for the s-algol external procedures. They are CLEARAREA, CLEARVDU, CURSORTO, ERRORVDU, IDSUB, IISUB, INPUTD, INPUTDK, INPUTDT, INPUTI, INPUTIK, INPUTIT, INPUTR, INPUTRK, INPUTRT, INPUTS, IRSUB, ISSUB, JULIN, JULOUT, OUTPUTI, OUTPUTR, SCROLL.

The first subroutine listed below (CLEARVDU) positions the cursor at a specific row (rows and columns numbered from 1) and clears from there to the end (row 24 column 80) of the vdu. The second subroutine (CURSORTO) positions the cursor at a specific row and column.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CLEARVDU.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
 SPECIAL-NAMES.
     SYMBOLIC CHARACTERS ASCII-ESCAPE IS 28.
*
* define the escape character to allow ANSI
*   standard cursor and screen addressing as
*   implemented on a VT100.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  CLEAR-VDU.
     03  FILLER          PIC X VALUE ASCII-ESCAPE.
     03  FILLER          PIC XXX VALUE "[0J".
 01  COLUMN             PIC 99 VALUE 1.
*
 LINKAGE SECTION.
 01  ROW                 PIC 99.
 PROCEDURE DIVISION USING ROW.
 CONTROLX SECTION.
 C00.
     CALL "CURSORTO" USING ROW, COLUMN.
     DISPLAY CLEAR-VDU WITH NO ADVANCING.
 C99.
     EXIT PROGRAM.


 IDENTIFICATION DIVISION.
 PROGRAM-ID. CURSORTO.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
 SPECIAL-NAMES.
     SYMBOLIC CHARACTERS ASCII-ESCAPE IS 28.
*
* define the escape character to allow ANSI
*   standard cursor and screen addressing as
*   implemented on a VT100.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ROW-COL.
     03  FILLER          PIC X VALUE ASCII-ESCAPE.
```

```
    03  FILLER          PIC X VALUE "[".
    03  ROW             PIC 99.
    03  FILLER          PIC X VALUE ";".
    03  COLUMN          PIC 99.
    03  FILLER          PIC X VALUE "H".
LINKAGE SECTION.
01  L-ROW               PIC 99.
01  L-COL               PIC 99.
*
PROCEDURE DIVISION USING L-ROW, L-COL.
CONTROLX SECTION.
C00.
    MOVE L-ROW TO ROW.
    MOVE L-COL TO COLUMN.
    DISPLAY ROW-COL WITH NO ADVANCING.
C99.
    EXIT PROGRAM.
```

This subroutine is used to input and verify a date at a specific row and column and within a specified range. It is used by three routines (i) INPUTD - to input a date (ii) INPUTDK - to input a date as a key (allowing '+' and c/r) (iii) INPUTDT - to input a date as the first field in a data entry detail line if there are a variable number of this record type in the data entry batch (allows c/r).

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. IDSUB.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ST-SIZE              PIC 99 COMP VALUE 6.
 01  LOWER-I           PIC 9(18)-.
 01  UPPER-I           PIC 9(18)-.
 01  W-JULIAN-DATE-WORK-AREA.
     03  W-JD-DATE              PIC 9(6).
     03  FILLER REDEFINES W-JD-DATE.
         05  W-J-DAY            PIC 99.
         05  W-J-MONTH          PIC 99.
         05  W-J-YEAR           PIC 99.
     03  W-J-JULIAN             PIC 9(6).
 01  ERROR-D.
     03  FILLER           PIC X(28) VALUE
         "Please enter a date between ".
     03  ED-LOWER         PIC 9(6).
     03  FILLER           PIC X(5) VALUE " and ".
     03  ED-UPPER         PIC 9(6).
 01  ERROR-MESSAGE        PIC X(78).
 01  INPUT-I           PIC S9(18) COMP.
 01  LOWER-J           PIC 9(6) COMP.
 01  UPPER-J           PIC 9(6) COMP.
 LINKAGE SECTION.
 01  INPUT-D           PIC 9(6) COMP.
 01  LOWER-D           PIC 9(6) COMP.
 01  UPPER-D           PIC 9(6) COMP.
 01  KEY-INPUT              PIC X.
 01  ROW               PIC 99.
 01  COLUMN           PIC 99.
 01  INPUT-S             PIC X(60).

*
 PROCEDURE DIVISION USING ROW, COLUMN, LOWER-D, UPPER-D,
     INPUT-D, INPUT-S, KEY-INPUT.
 CONTROLX SECTION.
 IND00.
     MOVE LOWER-D TO ED-LOWER.
     MOVE UPPER-D TO ED-UPPER.
```

```
    IF LOWER-D > ZERO
        MOVE LOWER-D TO W-JD-DATE
        CALL "JULOUT" USING W-J-JULIAN, W-J-DAY, W-J-MONTH,
        W-J-YEAR
        MOVE W-J-JULIAN TO LOWER-J
    ELSE
        MOVE ZERO TO LOWER-J.
    MOVE UPPER-D TO W-JD-DATE.
    CALL "JULOUT" USING W-J-JULIAN, W-J-DAY, W-J-MONTH, W-J-YEAR.
    MOVE W-J-JULIAN TO UPPER-J.
    MOVE ZERO TO LOWER-I.
    MOVE 999999 TO UPPER-I.
IND05.
    CALL "IISUB" USING ROW, COLUMN, LOWER-I, UPPER-I,
        INPUT-I, INPUT-S, KEY-INPUT.
    IF KEY-INPUT = "Y" AND (INPUT-S = "+" OR SPACES)
        GO TO IND99.
    IF KEY-INPUT = "T" AND INPUT-S = SPACES
        GO TO IND99.
    MOVE INPUT-I TO W-JD-DATE INPUT-D.
    IF INPUT-I = ZERO AND LOWER-D = ZERO
        MOVE ZERO TO W-J-JULIAN
        GO TO IND99.
    CALL "JULOUT" USING W-J-JULIAN, W-J-DAY, W-J-MONTH, W-J-YEAR.
    CALL "JULIN" USING W-J-JULIAN, W-J-DAY, W-J-MONTH, W-J-YEAR.
    IF W-J-JULIAN = ZERO OR W-JD-DATE NOT = INPUT-I
        MOVE "That's not a valid date" TO ERROR-MESSAGE
        GO TO IND90.
    IF W-J-JULIAN < LOWER-J OR W-J-JULIAN > UPPER-J
        MOVE ERROR-D TO ERROR-MESSAGE
        GO TO IND90.
    GO TO IND99.
IND90.
    CALL "ERRORVDU" USING ROW, COLUMN, ST-SIZE, ERROR-MESSAGE,
         INPUT-S.
    GO TO IND05.
IND99.
    EXIT PROGRAM.
```

This is the reciprocal of the s-algol external julian.in (see chapter 3.2 above). Given a date in DDMMYY format this routine will convert it into an integer number of days from 1/1/1950.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. JULOUT.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  FILLER.
     03  W-JD-DATE              PIC 9(6).
     03  FILLER REDEFINES W-JD-DATE.
         05  W-J-DAY            PIC 99.
         88  VALID-DAY VALUES 1 THRU 31.
         05  W-J-MONTH          PIC 99.
         88  VALID-MONTH VALUES 1 THRU 12.
         88  DAYS-31 VALUES  1 3 5 7 8 10 12.
         88  DAYS-30 VALUES  4 6 9 11.
         05  W-J-YEAR           PIC 99.
         88  VALID-YEAR VALUES 00 THRU 99.
 01  W-J-MONTH-WORK         PIC 999 COMP.
     88  DAYSW-31 VALUES  1 3 5 7 8 10 12.
     88  DAYSW-30 VALUES  4 6 9 11.
 01  W-J-YEAR-WORK          PIC 999 COMP.
 01  W-J-WORK               PIC 9(6) COMP.
 01  W-J-DAY-WORK           PIC 9(6) COMP.
 01  W-J-REM1               PIC 99 COMP.
 01  W-J-REM2               PIC 999 COMP.
 01  W-J-FEB                PIC 999 COMP.
 LINKAGE SECTION.
 01  W-J-JULIAN             PIC 9(6).
 01  W-J-D                  PIC 99.
 01  W-J-M                  PIC 99.
 01  W-J-Y                  PIC 99.
*
 PROCEDURE DIVISION USING W-J-JULIAN, W-J-D, W-J-M, W-J-Y.
 CONTROLX SECTION.
*
*****************************************************************
*                                                               *
*    W-J-DAY, W-J-MONTH, W-J-YEAR are converted to a julian date *
*    in W-J-JULIAN from a base of 1/1/1950.                     *
*                                                               *
*****************************************************************
 J000.
     MOVE W-J-D TO W-J-DAY.
     MOVE W-J-M TO W-J-MONTH.
     MOVE W-J-Y TO W-J-YEAR.
     IF VALID-YEAR AND VALID-MONTH AND VALID-DAY AND W-JD-DATE
         NUMERIC
         NEXT  SENTENCE
```

```
    ELSE
        MOVE ZERO TO W-J-JULIAN
        GO TO J099.
    IF W-J-YEAR < 50
        MOVE 18262 TO W-J-JULIAN
        MOVE ZERO TO W-J-YEAR-WORK
    ELSE
        MOVE 50 TO W-J-YEAR-WORK
        MOVE ZERO TO W-J-JULIAN.
J005.
    IF W-J-YEAR = W-J-YEAR-WORK
        GO TO J010.
    DIVIDE W-J-YEAR-WORK BY 4 GIVING W-J-WORK.
    MULTIPLY 4 BY W-J-WORK.
    SUBTRACT W-J-WORK FROM W-J-YEAR-WORK GIVING W-J-REM1.
    IF W-J-REM1 = ZERO AND W-J-YEAR-WORK NOT = ZERO
        ADD 366 TO W-J-JULIAN
    ELSE
        ADD 365 TO W-J-JULIAN.
    ADD 1 TO W-J-YEAR-WORK.
    GO TO J005.
J010.
    DIVIDE W-J-YEAR BY 4 GIVING W-J-WORK.
    MULTIPLY 4 BY W-J-WORK.
    SUBTRACT W-J-WORK FROM W-J-YEAR GIVING W-J-REM1.
    IF W-J-REM1 = ZERO AND W-J-YEAR-WORK NOT = ZERO
        MOVE 29 TO W-J-FEB
    ELSE
        MOVE 28 TO W-J-FEB.
    MOVE 1 TO W-J-MONTH-WORK.
J015.
    IF W-J-MONTH = W-J-MONTH-WORK
        GO TO J020.
    IF DAYSW-31
        ADD 31 TO W-J-JULIAN
    ELSE
    IF DAYSW-30
        ADD 30 TO W-J-JULIAN
    ELSE
        ADD W-J-FEB TO W-J-JULIAN.
    ADD 1 TO W-J-MONTH-WORK.
    GO TO J015.
J020.
    ADD W-J-DAY TO W-J-JULIAN.
J099.
    EXIT PROGRAM.
```

This sets the scrolling area on the vdu for batch data entry and is also used to set it off after the end of a data entry program run.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SCROLL.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
 SPECIAL-NAMES.
     SYMBOLIC CHARACTERS ASCII-ESCAPE IS 28.
*
* define the escape character to allow ANSI
*   standard cursor and screen addressing as
*   implemented on a VT100
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  SCROLL-ON.
     03  FILLER          PIC X VALUE ASCII-ESCAPE.
     03  FILLER          PIC X VALUE "[".
     03  SCROLL-FROM     PIC 99.
     03  FILLER          PIC X VALUE ";".
     03  SCROLL-TO       PIC 99.
     03  FILLER          PIC X VALUE "r".
 LINKAGE SECTION.
 01  ROW-FROM            PIC 99.
 01  ROW-TO              PIC 99.
*
 PROCEDURE DIVISION USING ROW-FROM, ROW-TO.
 CONTROLX SECTION.
 C00.
     MOVE ROW-FROM TO SCROLL-FROM.
     MOVE ROW-TO TO SCROLL-TO.
     DISPLAY SCROLL-ON WITH NO ADVANCING.
 C99.
     EXIT PROGRAM.
```

Conclusions

No-one has ever doubted the beauty of the algols. To extend the metaphor of the introduction (page 1 above) they are truly the Cinderallas compared to the ugly sisters and commercial bastions of COBOL and BASIC.

s-algol exhibits all that is good in language design.

Principally, the program structure is so complete that few logic errors find their way into the code. Most errors that bedevil standard languages are excluded by the necessity of completeness of logic. This is inextricably bound in with the flexibility of the language's facilities; for each piece of code of the same type to be interchangeable the program is composed of interchangeable logical units.

The resulting lack of rules removes major restrictions, giving the programmer a power that is both simple to use and difficult to misuse.

Secondly, the facilities supplied represent a clean sweep with a new broom, including what is good in old languages (e.g. 'if ... then ... else ...' - of course, an algol concept) and facilities new to most commercial programmers (e.g. 'case ...').

In particular a very honourable mention must be made of the structure and pointer facilities. These allow the programmer to build up core data structures at run time without the necessity of defining all possible types at their maximum sizes. In addition these can be linked together thus reducing the requirement for duplicate data in core.

It should be noted that these are very powerful facilities that take time to grasp initially and prove more powerful and useful as the programmer continues in the language. In addition these are the points that the programmer quickly loses his cutting edge on when he resorts to the hammer and chisel of COBOL or BASIC and subsequently returns to s-algol.

But s-algol would be dismissed in commerce as, despite its obvious power, it lacks some of the basic requirements of commercial programming. However, this is not the end of the story, a successor, ps-algol, has extended and improved on s-algol to provide all the facilities required by the commercial programmer.

I do not doubt that ps-algol could outperform the extant commercial third generation languages for speed of coding, correctness of code and ease of subsequent maintenance. The language uses less code than COBOL, is more readable and subsequently faster to code in with far fewer logic errors being accepted by the compiler.

Usually with an algol a dose of what commercial people such as myself call 'reality' would now take over. The illogical but one-sided story of portability of COBOL/BASIC/FORTRAN code, programming knowledge of these languages, etc. Even here the limiting factor on s-algol and ps-algol is the lack of a broad machine base as the language is almost 100% portable across its implemented machine base. COBOL is portable across a much larger machine base but with far less ease. BASIC is not really portable, requiring almost complete rewrites on system conversion. But these languages are well known and used.

COBOL is no longer the end of the commercial language story. For some time newer concept languages, the fourth generation, have been gradually consolidating their position in the commercial programming market. These languages and tools do much of the repetitive coding for the programmer, producing documentation and generally speeding up systems development by hundreds of percentage points. Thus commerce is accepting gradually that Emperor COBOL has no clothes on. However, any advantages ps-algol and s-algol have proved over the third generation languages are rapidly lost in the fourth generation.

The fourth generation is a different market than the third generation. The third generation consists principally of COBOL and BASIC (in commerce) and FORTRAN (in industry). The rest come a long way down the field as also-rans, with the algols. However, the fourth generation lacks a single approach to the market, some facilities are languages, some are generators and some have almost no compilation requirement. Much of the fourth generation is also badly designed and tested.

If the fourth generation as it stands did everything that the marketers claim then there would be no requirement for the third generation or for that matter for further development. Of course the claims are at best exaggerated. The fourth generation tools provide some of the facilities that could be automated but lack an overall philosophy to provide a 'better mousetrap'. Thus each tool has good points and bad points that its opposition will quickly leap on.

There is still a large gap for a better tool that has the flexibility to generally tackle the various aspects of commercial programming.

The algol and COBOL generators produced above (chapter 4) exhibited the possibility of producing complete logic structures and documentation from a very few parameters. A possibility identified by surprisingly few fourth generation products, most only making the elementary deduction that commercial data processing consists of data in and data out.

But there is far to go, much of which is identifiable. The fourth generation is an open-ended one as regards the possibility of acceptance of new and better products.

There are three identified and related stages of correlation that together define a consistent system:

1: field consistency - the data dictionaries exhibit the need of keeping data type definitions consistent. Thus the database manager can be assured that each field will have the same characteristics no matter where it occurs on the database.

2: record consistency - this could be taken a step further by linking the separate occurrences of this named field to its associated records in a deletion/addition chain: do not add a record of type 'A' unless a record of type 'B' already exists and do not delete a record of type 'B' if a record of type 'A' exists.

This expands the concept of automatic field consistency to automatic record consistency, hitherto approached, if at all, from the entirely separate database design angle alone.

3: file consistency - further, the COBOL data entry generator (tpg) produced above exhibited the need to go one step further to file consistency. The generator's data processing program only produced an audit of the data without actually processing it. With the files

defined (in stages 1 and 2 above) and data flows from these files in terms of records and fields defined within the system then the parameters required to input and process data should then also have been defined.

This was partially exhibited by combining the parameters required to produce both a batch data entry program and the corresponding data processing program into a single file (the data processing parameters were a subset of the data entry parameters). Thus two very different programs were produced from an associated set of parameters and were therefore consistent with each other and within the system.

The possibility exists for a complete suite to be defined entirely at parameter level with complete consistency of data storage.

Doubtless unique codings may well be required, but many of these may prove to be figments of the analyst's imagination as others proved to be in the file maintenance suite. This is probably the reason that no current fourth generation tools have tackled this 'ideal' solution. The commercial data processing market is short-sighted in terms of the time it is willing to wait before profits appear. Hence the fourth generation facilities have a limited amount of thought and testing before they are marketed and subsequently enhanced. The above outlined facility would require a long term commitment before anything saleable may occur.

As mentioned above (chapter 4 section 1) the non-compilation-bound facility could certainly lead the market. But with a logical system design core identified as outlined above, such a tool could produce COBOL or even algol, as some parts of the market may prefer.

In use, the St Andrews family of algols produced general rules in relation to the extant commercial languages I have used (COBOL, BASIC, RPG and VISTA): (i) their compiled code is cleaner (ii) they can be clean compiled quicker (iii) they are more readable (iv) they require less source code (except in comparison to the 4gl VISTA). With the advent of the fourth generation with no preconceptions against the algol family, there is an opportunity for ps-algol. With its coding benefits and uniquely powerful database handling the opportunity exists to produce a genuine commercial programming market leader.

I cannot see an algol, even such as s-algol or ps-algol, making a great impact on the ageing third generation. However, there are real areas for improvement in the fourth generation with an open-minded approach being evidenced by the commercial community. This is an area where the language could be heading.

APPENDIX A

IFS MAILSHOT

Abtech Business Systems Ltd

BIS Applied Systems Ltd.

Background Computer Systems Ltd.

Beta Systems

Buick Computer Services

CAP Group Ltd.

CSC OK Computer Services Co. Ltd.

Calapine Ltd.

Cambridge Consultants Ltd.

Cocking and Drury Ltd.

Compiler and Language Consultancy Ltd.

Computer Systems International Ltd.

D.M. England and Partners Ltd.

DP Support Services Ltd.

Dacris Computer Consultants

Data Highways Ltd.

Datex Micros

David Livingstone Consultants Ltd.

Dillon Computing Management Ltd.

Dyadic Systems Ltd.

Dynamic Software Services

Erlebach Engineering Ltd.

Eurolink Computer Services Ltd.

Evets Consultancy Ltd.

F International Ltd.

Ferranti Computer Systems (Cwmbran Software)

Fleet Electronics Consultancy

Gareth Morgan Computer Services

Graffcom Systems Ltd.

H Programming & Systems Ltd.

HLW Computer Services Ltd.

HSOP Text Processing Services Ltd.

Haverley Systems Europe Ltd.

Heritage Computer Services Ltd.

High Integrity Systems Ltd.

Hill, Price, Davison Ltd.

Holland Automatic International

IPL Information Processing Ltd.

Impax Computer Services Ltd.

Inbucon Ltd.

Industrial and Commercial

Informatics Consulting Ltd.

Informex-London Ltd.

Intelligence (IRL) Ltd.

Inter-City Programming Support Ltd.

John Bell Computer Services Limited

KPG Computer Systems Ltd.

Knight COmputer Services Ltd.

Leasco Software Ltd.

M R Dataflow

MMT Computing Ltd.

MSS Computer and Business Consultancy

Manex Management Ltd.

Mapp Computer Facilities Ltd.

Marcol Computer Services Ltd.

Martin Brampton Software Ltd.

Michael Johnston & Co.

Microware Computer Systems Ltd.

Micronology Ltd.

Micro Automation Computing Ltd.

Micro Scope Ltd.

Millenium Professional and Technical

Mills and Allen Communications Ltd.

Milspec Systems Ltd.

Montreal Associates (Systems) Ltd.

Mountford & Laxon Co. Ltd.

National Data Processing Service

Nine Tiles Information Handling Ltd.

OCR Systems

Oestreicher, Strauss and Warwick

Open COmputer Services

PACTEL

PIM/Inicom Ltd.

Pioneer Computer Systems Ltd.

Prestedge Consultants Ltd.

Quantum Science Corporation

RAS Systems Design Ltd.

Real Time Developments Ltd.

Riverside Consultants Ltd.

SPC Systems Ltd.

SPL International

Science Systems Ltd.

Servelec Computer Systems Ltd.

Shade (Computer Services) Ltd.

Software Expertise Ltd.

Software Ireland Ltd.

Software and Programming Consultants

South Eastern Computer Services

Spectronics Microsystems Ltd.

Synchro Systems Ltd.

Systems Consultancy

Systems Designers Ltd.

Systems Consultants Ltd.

T & ACS

Total Systems Ltd.

Triad Computing Systems Ltd.

Twenty First Computer Systems Ltd.

Tygoze Ltd.

United Computer Resources

Venture Computing Ltd.

Warren Point Ltd.

Wintec Ltd.

Wooton, Jeffreys & Partners

Xoren Computing Ltd.

Y-Ard Ltd.

APPENDIX B

CHARLES LETTS FINAL CORRESPONDENCE

Charles Letts (Scotland) Ltd.

Registered Office:
Thornybank Industrial Estate
Dalkeith, Midlothian EH22 2NE

Telephone 031663 1971
Telegrams: 'Diarists' Dalkeith
Telex: 72348
Registered No. 7315 Edinburgh

Mr J Sutherland
Admin./Library Computer Unit
Old Union
North Street
University of St Andrews
St Andrews
KY16 9AN

Our Ref Your Ref Date 30 August 1983

Dear John

Thank you for your letter dated 25th August 1983. Naturally I was disappointed to learn of your difficulty in converting our Warehouse Distribution System from Business Basic to S-Algol. I am sure my colleagues and indeed our mutual friends from Fraser Williams would share my disappointment.

I would, however, stress that although the outcome of your project proved fruitless, the experience, knowledge and understanding gained will be of significant benefit - at the very least both Fraser Williams and Charles Letts are aware of these difficulties if approached by a potential S-Algol customer.

Our success with the Business Basic version of the Warehouse Distribution System can be directly attributed to the efficient method of file access, that of Index Sequential access where data can be written to or read from a data record given one or more index keys. Since the file organisation is based on the data record rather than the data block, Index Sequential file access offers substantial flexibility.

This technique is critical with the Stock Location File where six different indexes are required to access the data - Indeed without this method of file organisation

Mr J. Sutherland

30 August 1983

I would question the efficiency, the flexibility and the commercial acceptability of the system. In view of the fact that the language S-Algol does not support this method of file organisation I would agree with your decision to drop the project.

In closing I offer my appreciation of your efforts over the last few months and wish you every success for your future plans.

Yours sincerely
FOR CHARLES LETTS (SCOTLAND) LTD.

C. S. DEWAR
Systems Analyst

Fraser
Williams
(Scotland)
Limited

COMPUTER SERVICES

Stock Exchange House
69 St. George's Place
Glasgow G2 1QY
Telephone 041-226 3864

KMMcP/IH

Mr J N Sutherland
Administrative & Library Computer Unit
The University
Old Union
North Street
ST ANDREWS .

21st September 1983

Dear John,

Charles Letts Warehousing System

I was sorry to hear of your failure to convert the above system into S-Algol. I also agree that it would be impossible without the use of our indexed-sequential filing system. As you know we would never contemplate purchasing a language or developing a system in a language that did not provide this facility, for the very obvious reasons of lack of flexibility, and escalation of development times and therefore costs.

However, I would like to wish you all the best in your new project for the Byre Theatre.

Do keep in touch.

Yours sincerely,

K M McPartland
General Manager

APPENDIX C

4GL MAILSHOT

ADR Limited

Access Technology Limited

Adds (UK) Ltd.

Adrem Systems plc

Amethyst Computer Resources

Ashton Tate

Assyst (UK) Computer Services Ltd.

Burroughs Machines Limited

Cincom Systems (UK) Ltd.

Clasma Systems Limited

Cognos

Compsoft plc

Computer Modelling Limited

Control Data Limited

Cullinet Software Ltd.

DMW Group Europe

Data Design Computer Services

Delta Software Tools

Doric Computer Systems

E.S.I.

Equinox Computers

Expert Systems International

Fee Software Products Ltd.

Hammond Software UK

Honeywell Information Systems

Hoskyns Group Limited

ICL

Information Builders (UK) Ltd.

Informatics General (UK) Ltd.

Infosys Ltd.

Intercom Data Systems Limited

Interactive Computer Systems Ltd.

J.P.Y Associates Ltd.

Jenson Computer Systems Limited

Ki Computer Services Ltd.

Logica UK Ltd.

MDN (PRO-IV) Limited

MS Associates Ltd.

MSA (Management Science America) Ltd.

Majic Software (C.C. Limited)

Mathematica Products Group

Microdata Information Systems Ltd.

Micro Focus Ltd.

Morino Associates

NCR Ltd.

Oracle UK

Pansophic Systems (UK) Ltd.

Pioneer Computer Systems

Prime Computer (UK) Ltd.

SAS Software Limited

SIA Computer Services

Sapphire Systems Limited

Savant Enterprises

Savant Enterprises

Software AG of the United Kingdom ltd

Software Marketing International Ltd.

Sperry Limited

Sphinx Limited

Systime Limited

Tamsys Limited

Telesystems

Tetra Business Systems Ltd.

The Bristol Software Factory

Thorn EMI COmputer Software

Tubs Software Limited

Unit-C

Bibliography

"1 2 3 4th Generation Language Software"; Business Software Review UK Edition; International Computer Programs, Inc., 6-8 Cole Street, London SE1 4YH; pp28-32; february 1986.

Accounts Department Data Entry Computer Manual; 1.1; Sutherland, J.N.; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; February 1985

ADABAS (VMS) Introduction; ADV-100-010; Software AG, Darmstadt, W. Germany; May 11, 1984.

ADR/IDEAL; SI1G-00-30; Applied Data Research, INC., Route 206 and Orchard Road, CN-8, Princeton, NJ 08540-9936, U.S.A.; July 1985

ADR/DATACOM/DB; DB1G-00-20; Applied Data Research, INC., Route 206 and Orchard Road, CN-8, Princeton, NJ 08540-9936, U.S.A.; July 1985

ADR/DATADICTIONARY; DD1G-00-20; Applied Data Research, INC., Route 206 and Orchard Road, CN-8, Princeton, NJ 08540-9936, U.S.A.; January 1986

ADS/ONLINE summary description; Cullinet Software Inc., 400 Blue Hill Drive, Westwood, MA, U.S.A.; 1985

American National Standard COBOL Compiler ; AH30, Rev 1 ; Honeywell Corporation ; Minneapolis, U.S.A. ; December, 1973

An Introduction to SYSTEL; 007-2202/C; Systime Limited, Concours Computer Centre, 432 Dewsbury Road, Leeds; 1982

Atkinson, M.P., et al; 1984; ps-algol Reference Manual ; University of Edinburgh Department of Computing Science and University of St. Andrews Department of Computational Science; The University of St. Andrews, Department of Computational Science ; 17/1/84

Business BASIC Directory (AOS/RDOS/DOS) ; 093-000226 ; Data General Corporation ; Westboro, Massachusetts, U.S.A.

Business BASIC Reference Manual ; 093-000137-03 ; Data General Corporation ; Westboro, Massachusetts, U.S.A. ; 1978

Cash Office Data Entry Manual; 1.0; Sutherland, J.N.; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; July 1985

Chai W.A. and Chai W.H. ; Programming Standard COBOL ; Academic Press (London) Inc. ; 24/28 Oval Road, London NW1, England ; 1976

Cole A.J. and Morrison R.; An introduction to s-algol programming; CS/80/1 ; University of St. Andrews, Department of Computational Science ; 1980

DATA FLEX; DATAFLEX (information Management) Services Ltd., 16 Anning Street, New Inn Yard, London EC2A 3HB

Data Master, an introduction; Sapphire Systems Limited, 180 Cranbrook Road, Ilford, Essex.

Draft Proposed American Standard Database Language, SQL; American National Standards Institute, Inc.; February 1985.

Eve J. ; Algol W Programming Manual ; 01JULY72 ; University of Newcastle Upon Tyne ; Claremont Tower, Newcastle Upon Tyne, England NE1 7RU ; 1st July , 1972

FIDEL; DN40008.0582; Information Builders, Inc.,1250 Broadway, New York, N.Y. 100001, U.S.A.

FOCUS the total English language software resource; DN40005.0582; Information Builders, Inc.,1250 Broadway, New York, N.Y. 100001,

U.S.A.

Grant ; The Computer Users' Yearbook ; C.U.Y.B. Publications Ltd. ; 430-432 Holdenhurst Road, Bournmouth BH8 9AA, England ; 1982

Hanson, Owen; Design of Computer Data Files; Pitman Books Ltd., 128 Long Avenue, London WC2 9AN; 1982.

I.B.M. O.S. Full American National Standard COBOL ; Rev 4, GC28-6396-4 ; I.B.M. World Trade Corporation ; 821 United Nations Plaza, New York 10017, U.S.A.

IBM System/34 RPG II Reference Manual ; SC21-7667-1 ; International Business Machines Corporation ; General Business Group/International, 44 South Broadway, White Plains, New York 10601, U.S.A. ; 1978

INFO for greater productivity; Doric Computer Systems, Doric House, 23 Woodford Road, Watford WD1 1PB.

Interactive COBOL Programmer's Reference ; 045-011-00 ; Data General Corporation ; Westboro, Massachusetts 01581 , U.S.A. ; 1977

Introduction to INFO-Text; Henco Software, Inc., 100 Fifth Avenue, Walthamn Mass 02154, U.S.A.;1984

Introduction to VAX-11 Record Management Services; AA-D024D-TE; Digital Equipment Corporation, Maynard, Massachusetts; May 1982

Jones, Russell; 'PRO-IV: aming to kick dp staff into the eighties'; IBM Computer Today; 9/11/85

Kelly, J. & Sutherland J.N.; Computer Operating Instruction Manual for the Warehouse Distribution system; Fraser Williams (Scotland) Ltd., Stock Exchange House, 69 St. George's Place, Glasgow G2 1QY; 1981

Kelly, J. & Sutherland, J.N.; Program Specifications for the Warehouse Distribution system; Fraser Williams (Scotland) Ltd., Stock Exchange House, 69 St. George's Place, Glasgow G2 1QY; 1981

Kelly, John; System Specification for the Warehouse Distribution system; Fraser Williams (Scotland) Ltd., Stock Exchange House, 69 St. George's Place, Glasgow G2 1QY; 1980

MICS Evaluation Series; Morino Associates, Inc., 8615 Westwood Centre Drive, Vienna, Virginia 22180-2215, U.S.A.; 1985

MIMER the software machine; Savant, 2 New Street, Carnforth, Lancs. LA5 9BX, England

MODUS Transaction Processing; Computer Technology Limited; 30/000015B; CTL Limited, Eason Road, Hemel Hempstead, Hertfordshire HP2 7LB; July 1978

Morrison R.; S-algol Reference Manual; CS/79/1; University of St. Andrews, Department of Computational Science ; 1979

NATURAL the proven 4th generation technology, concepts and facilities; NAT-210-005; Software AG, Darmstadt, Federal Republic of Germany; August 1985.

NOMAD2 an overview; D-2B10K; The Dun & Bradstreet Corporation, 187 Danbury Road, Wilton, CT 06897, U.S.A.; 10/1984

NOMAD2 the software; V/N 100071;The Dun & Bradstreet Corporation, 187 Danbury Road, Wilton, CT 06897, U.S.A.; 11/1984

Prime INFORMATION Made SIMPLE; Prime Computer, Inc., Natick, Massachusetts, U.S.A.

Print Invoices for Residence Fees; Staddle, J.; University of St.

Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; April 1981

PRO-IV an overview; PRO Computer Sciences, Inc., 23181 Verdugo Drive, Suite 103A, Laguna Hills, Calif 93653, U.S.A.; August 1985

PRO-IV User's Guide; version 1.30; PRO Computer Sciences, Inc., 23181 Verdugo Drive, Suite 103A, Laguna Hills, Calif 93653, U.S.A.; August 1985

Programming the 1900 Series in COBOL ; International Computers Limited ; ICL Training, ICL Beaumont, Old Windsor, Berkshire, England ; 1977

Proposals for Payment of Creditors System; Staddle, J. & Christie, A.D.G.; FS402; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; December 1977

Powerhouse Primer 1; PHVXP01; Cognos Incorporated, 275 Slater Street, 10th Floor, Ottowa, Canada K1P 5H9; April 1984

Robinson, Barry; 'SQL is not enough'; SIR Incorporated, 5215 Old Orchard Road, Suite 800, Skokie, IL 60077, U.S.A.; October 1985

Software Development with DELTA; MA 216; Delta Software Technologie AG, Bahnstrasse 5, CH-8603 Scherzenbach, Switzerland; June 1985

Speedware high-performance fourth-generation productivity software; Infocentre, 9 Vine Lane, Tower Bridge, London, England DE1 2JQ.

Ullman, Jeffrey D.; Principles of Database Systems; 1980; Computer Science Press; 11 Taft Ct., Rockville, Maryland 20850, U.S.A.

ULTRA; MB-00410M-11/83; Cincom Systems, 2300 Montana Avenue, Cincinnati, OH 45211, U.S.A.

User 11; Fact Sheet No. 8; Jenson Computer Systems Limited, 30 Queen Square, Bristol BS1 4ND.

User 11 Software Data Sheet; Pioneer Computer Systems Ltd., 4 Albion Place, Northampton NN1 1UD, UK

User's Manual for Video Terminal CIT-101e; preliminary edition; CIE Terminals, Citoh Electronics; October 1983.

VAX-11 BASIC Language Reference Manual ; AA-H867A-TE ; Digital Equipment Corporation ; Maynard, Massachusetts, U.S.A.; 1980

VAX-11 Record Management Services Reference Manual; AA-D031D-TE; Digital Equipment Corporation ; Maynard, Massachusetts, U.S.A.; May 1982

VAX COBOL Language Reference Manual; AA-H631C-TE; Digital Equipment Corporation ; Maynard, Massachusetts, U.S.A.; October 1984

VISTA Base Module Details; Vista Computer Services Ltd., 35 Soho Square, London W1; 1st October 1982

VISTA Specification Language Commands; Vista Computer Services Ltd., 35 Soho Square, London W1; 1st October 1982

VISTA Specification Language Verbs; Vista Computer Services Ltd., 35 Soho Square, London W1; 1st October 1982

'Voyage of Discovery'; Computing Futures Limited

VT100 User's Guide; EK-VT100-UG-002; Digital Equipment Corporation ; Maynard, Massachusetts, U.S.A.; Jan 1979.

'What changing to a computer can mean'; Storage Handling Distribution; May 1982; pp30-31.

Whiteside, David; 'Coping with the shortage'; Datamation Magazine, Dun and Bradstreet; 1985

Whiteside, David; 'Selection starts for true fourth generation'; DEC User, EMAP Business & Computer Publications Ltd., 67 Clerkenwell Road, London EC1R 5BH; May 1985

References

1. "1 2 3 4th Generation Language Software"; Business Software Review UK Edition; International Computer Programs, Inc., 6-8 Cole Street, London SE1 4YH; pp28-32; February 1986.

2. Accounts Department Data Entry Computer Manual; 1.1; Sutherland, J.N.; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; February 1985

3. American National Standard COBOL Compiler ; AH30, Rev 1 ; Honeywell Corporation ; Minneapolis, U.S.A. ; December, 1973

4. An Introduction to SYSTEL; 007-2202/C; Systime Limited, Concours Computer Centre, 432 Dewsbury Road, Leeds; 1982

5. Atkinson, M.P., et al; 1984; ps-algol Reference Manual ; University of Edinburgh Department of Computing Science and University of St. Andrews Department of Computational Science; The University of St. Andrews, Department of Computational Science ; 17/1/84

6. Business BASIC Directory (AOS/RDOS/DOS) ; 093-000226 ; Data General Corporation ; Westboro, Massachusetts, U.S.A.

7. Business BASIC Reference Manual ; 093-000137-03 ; Data General Corporation ; Westboro, Massachusetts, U.S.A. ; 1978

8. Cash Office Data Entry Manual; 1.0; Sutherland, J.N.; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; July 1985

9. Chai W.A. and Chai W.H. ; Programming Standard COBOL ; Academic Press (London) Inc. ; 24/28 Oval Road, London NW1, England ; 1976

10. Draft Proposed American Standard Database Language, SQL; American

National Standards Institute, Inc.; February 1985.

11. Eve J. ; Algol W Programming Manual ; 01JULY72 ; University of Newcastle Upon Tyne ; Claremont Tower, Newcastle Upon Tyne, England NE1 7RU ; 1st July , 1972

12. Grant ; The Computer Users' Yearbook ; C.U.Y.B. Publications Ltd. ; 430-432 Holdenhurst Road, Bournmouth BH8 9AA, England ; 1982

13. Hanson, Owen; Design of Computer Data Files; Pitman Books Ltd., 128 Long Avenue, London WC2 9AN; 1982.

14. I.B.M. O.S. Full American National Standard COBOL ; Rev 4, GC28-6396-4 ; I.B.M. World Trade Corporation ; 821 United Nations Plaza, New York 10017, U.S.A.

15. IBM System/34 RPG II Reference Manual ; SC21-7667-1 ; International Business Machines Corporation ; General Business Group/International, 44 South Broadway, White Plains, New York 10601, U.S.A. ; 1978

16. Interactive COBOL Programmer's Reference ; 045-011-00 ; Data General Corporation ; Westboro, Massachusetts 01581 , U.S.A. ; 1977

17. Jones, Russell; 'PRO-IV: aming to kick dp staff into the eighties'; IBM Computer Today; 9/11/85

18. Kelly, J. & Sutherland J.N.; Computer Operating Instruction Manual for the Warehouse Distribution system; Fraser Williams (Scotland) Ltd., Stock Exchange House, 69 St. George's Place, Glasgow G2 1QY; 1981

19. Kelly, J. & Sutherland, J.N.; Program Specifications for the Warehouse Distribution system; Fraser Williams (Scotland) Ltd., Stock

Exchange House, 69 St. George's Place, Glasgow G2 1QY; 1981

20. Kelly, John; System Specification for the Warehouse Distribution system; Fraser Williams (Scotland) Ltd., Stock Exchange House, 69 St. George's Place, Glasgow G2 1QY; 1980

21. MODUS Transaction Processing; Computer Technology Limited; 30/000015B; CTL Limited, Eason Road, Hemel Hempstead, Hertfordshire HP2 71B; July 1978

22. Morrison R.; S-algol Reference Manual; CS/79/1 ;University of St. Andrews, Department of Computational Science ; 1979

23. NATURAL the proven 4th generation technology, concepts and facilities; NAT-210-005; Software AG, Darmstadt, Federal Republic of Germany; August 1985.

24. NOMAD2 the software; V/N 100071;The Dun & Bradstreet Corporation, 187 Danbury Road, Wilton, CT 06897, U.S.A.; 11/1984

25. Print Invoices for Residence Fees; Staddle, J.; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; April 1981

26. PRO-IV an overview; PRO Computer Sciences, Inc., 23181 Verdugo Drive, Suite 103A, Laguna Hills, Calif 93653, U.S.A.; August 1985

27. PRO-IV User's Guide; version 1.30; PRO Computer Sciences, Inc., 23181 Verdugo Drive, Suite 103A, Laguna Hills, Calif 93653, U.S.A.; August 1985

28. Programming the 1900 Series in COBOL ; International Computers Limited ; ICL Training, ICL Beaumont, Old Windsor, Berkshire, England ; 1977

29. Proposals for Payment of Creditors System; Staddle, J. & Christie, A.D.G.; FS402; University of St. Andrews, Administrative and Library Computer Unit, Old Union Building, North Street, St. Andrews; December 1977

30. Robinson, Barry; 'SQL is not enough'; SIR Incorporated, 5215 Old Orchard Road, Suite 800, Skokie, IL 60077, U.S.A.; October 1985

31. Software Development with DELTA; MA 216; Delta Software Technologie AG, Bahnstrasse 5, CH-8603 Scherzenbach, Switzerland; June 1985

32. Ullman, Jeffrey D.; Principles of Database Systems; 1980; Computer Science Press; 11 Taft Ct., Rockville, Maryland 20850, U.S.A.

33. User's Manual for Video Terminal CIT-101e; preliminary edition; CIE Terminals, Citoh Electronics; October 1983.

34. VAX-11 BASIC Language Reference Manual ; AA-H867A-TE ; Digital Equipment Corporation ; Maynard, Massachusetts, U.S.A.; 1980

35. VISTA Base Module Details; Vista Computer Services Ltd., 35 Soho Square, London W1; 1st October 1982

36. VISTA Specification Language Commands; Vista Computer Services Ltd., 35 Soho Square, London W1; 1st October 1982

37. VISTA Specification Language Verbs; Vista Computer Services Ltd., 35 Soho Square, London W1; 1st October 1982

38. 'Voyage of Discovery'; Computing Futures Limited

39. VT100 User's Guide; EK-VT100-UG-002; Digital Equipment Corporation ; Maynard, Massachusetts, U.S.A.; Jan 1979.

40. 'What changing to a computer can mean'; Storage Handling Distribution; May 1982; pp30-31.

41. Whiteside, David; 'Coping with the shortage'; Datamation Magazine, Dun and Bradstreet; 1985

42. Whiteside, David; 'Selection starts for true fourth generation'; DEC User, EMAP Business & Computer Publications Ltd., 67 Clerkenwell Road, London EC1R 5BH; May 1985