An Approach to Debugging

by Grant W. Fletcher of The Interface Group, Incorporated, and
Kathleen A. Sachara of The Haley Corporation

Abstract of the Paper

A significant amount of effort and, therefore, money is spent debugging systems during the development cycle and after installation.

An Approach to Debugging defines the issue in terms of development and post-installation situations, and then goes on to present a philosophical foundation upon which the matter may be approached to optimize system user/system development efforts.

An Approach to Debugging also presents, in a case study model, some tools and techniques that are applicable to COBOL programming and which may be tailored to other languages.

The objectives of the paper are to present a philosophical approach to debugging that may cause the audience to re-evaluate their own approach to the task, and to share some techniques that have been applied to the task.

Agenda of the Presentation

The What, Why, and When of Debugging.

It is always useful to define the topic at hand, and that is what is addressed by way of the introduction to the paper. The audience is given a definition of what the task is, and some explanation of why and when debugging becomes a necessity.

A Philosophical Foundation.

The success, or failure of a system may depend upon the ability of a system user and a system developer to co-operate during the development cycle and after installation, and that co-operation itself may be dependent upon the philosophy of each participant as it relates to the other. One philosophical approach to the task of debugging that attempts to optimize the user/developer relationship is presented.

Some Tools and Techniques.

The usefulness of knowledge is difficult to quantify until the knowledge itself is evidenced in some material form. The paper presents a case study of some tools and techniques that have been applied to debug an application, both during development and after installation.

The What, Why, and When of Debugging.

Software is usually in a perpetual state of evolution. It is very rare to find a computer programme that has been implemented, and that is not expected to require some modification work in the future. Ideally, we hope that any subsequent modifications to our own software are enhancements that are in response to new user requirements or procedures. Some modification, realistically, occurs in response to user problems encountered with an aspect of the design, coded logic, implementation environment, or manner in which the software is used. Debugging may be viewed as an analytical tool, a technique, or simply the process of addressing the requirements of the latter case.

Webster's defines the term debug as a verb meaning "to find and correct the defects, errors, malfunctioning parts, etc. in". More specifically applied to the job of computer programmers, this definition may be re-written to emphasize the reality that errors may be real or perceived, and that errors occur in computer systems. So, we may now define debugging as the task of finding and correcting the real or perceived errors in computer systems.

The errors that we are commonly required to correct in computer software include errors in design and coded logic, errors in installation environments, and errors related to the usage of the software in question. Their implication may be related to one or more of many factors.

Typically, errors will manifest themselves in the form of corrupted output from a particular programme. Unfortunately, in our more complex systems where data dependancies are numerous, an error may come to surface in a programme that is some number of process steps removed from the offending programme.

Also, in today's business environment, errors can manifest themselves in other ways. Many of our systems have evolved beyond the point of being simple procedural tasks that may be isolated in a few independent computer programmes. Because of the complexity of our systems, and often because of the complexity, necessary, or otherwise, of our computer programmes, user training and the usage of computer programmes themselves may be the source of perceived errors. Perceived errors are more difficult to address as they are often presented as errors relating to programme output. But the perception that an error exists arises because the user's expectation of the output from the programme is not what the programme is intended to produce. If the user's expectations are based upon incorrect assumptions, incomplete training, or improper procedural responses, then an error in user education or software usage may exist. It is as equally important to correct this type of error as those that are directly related to the work done by system developers.

A third manifestation of errors in computer programmes that is becoming more critical in today's business environment, and which should be of particular interest to programmers in an HP3000 (pre Spectrum) environment, is programme performance. Systems may become needlessly overburdened with tasks that may be optimized with better programme or data base design, and a hardware upgrade is simply not available, or not economical. In these situations, if not always, programme performance may be the evidence of errors within the programme's logic, the techniques employed to code the logic, or it may be

considered to be the error in itself.

Why errors exist is more ambiguous than the fact that errors do exist, but is of more importance to resolve. With the tools that are available to programmers and other system developers today, it is very easy to write and install computer programmes. But, few, if any of today's tools remove the potential for creating systems that include errors. So, since it is now possible to increase the number of computer programmes produced by one computer programmer during a particular period of time without decreasing the potential for error in each of those programmes being produced, one may argue that the by-product of database management systems, fourth generation languages, and other software advances is the creation of more software errors. This is certainly the case when the causes of errors are not addressed prior to the introduction of some of today's tools.

Errors may be introduced to the computer system early in the project life cycle. One of the first causes of errors in computer systems that eventually require debugging is the communication skills of those people involved with the initial design of a computer programme or system. Poorly communicated requirements, unclear or overly technical proposals, and other misunderstandings can result in systems being developed and implemented that never actually resolve the user's initial request.

Following the project life cycle, the next situation in which errors may be introduced is in the development phase, or during the coding of specific programmes and procedures. Syntax and logic related errors are a very common source of errors in programmes, both during development and after the software has been installed and used.

Another source of errors in programme code that is evidenced more and more today because our systems are often one node in a multi-hardware configuration is the way data is stored within the programme. Once again, communication, now of the intended use of the data, either by designer to programmer, or user to designer, may be the cause of errors detected after programme installation.

Errors that are attributable to programme performance are cases in which the design or technique of coding a particular programme is not the best, or when minor sections of code require inordinant amounts of CPU time to process. These errors can be difficult to detect without prior consideration during the design of the software, and can usually be credited to the level of expertise of the development staff.

Errors may also arise because programmes are developed and used in different account structures, operated under different versions of operating systems, or on completely different CPU's. These types of errors, typically, are the easiest to resolve in a systematic way as each difference between the two environments can be isolated and proven to be the offending factor, or not.

Misconceptions, and improper usage of software are usually the result of inadequate or unclear documentation, or other training factors. When programmes are designed, coded, and installed perfectly there is still the requirement that the programmes be used in the manner, and for the purpose which they were created. Documentation and training can be seen to be the cause of many perceived errors in computer systems. The responsibility to

provide a sufficient level of documentation or training for the user has to begin with the designers and developers of computer systems as they are most familiar with the intended usage of the programmes that they implement.

A Philosophical Foundation.

Whether tool, technique, or task, debugging is often the least scientific element of our function as computer programmers. It is common to hear debugging described as an art, or some other subjective process, and it is too often found that the task is delegated to our most junior staff and rationalized as an important learning experience for them to endure. Rarely are programmes or systems designed with any direct consideration for the fact that the project will undergo some level of debugging during it's life cycle, and that the implemented work from the project will itself be the subject of some future debugging, or at least investigation of it's processing or performance.

The reasons discussed for why and when debugging occurs have one common element, and that one element contributes to the ambiguity and subjectiveness found relative to the task. The ability of the user to communicate their requirement to the system designer is the first potential cause of errors in computer programmes. Then, the ability of the designer to communicate the intent and specification of the programme to the programmer, the ability of the programmer to describe and document the work that has been done, and the ability of the user to understand the use of the implemented programme all compound the potential for errors in the systems that we develop. Debugging becomes a difficult task because the interpretation of each participant in the project of what the project is addressing is rarely similar.

But, there is no good reason why debugging cannot be made easier with a different approach to the matter. Computer scientists are the rare exception within the scientific community who appear to assume that their work will be functional and error free on the first attempt, and rarely design for the possibility that is not. Most computer programmes are written under the philosophy that testing will isolate any and all errors, and that the errors that do occur are unique enough that they cannot be generalized. Perceived errors are rarely considered to be within the realm of responsibility of computer programmers, but usually considered to be problems that the users themselves must resolve.

Approaching system development with the idea that debugging is a design issue, rather than an ad hoc activity to be considered if and when problems arise, causes programmes to be written that are easier to debug. Designing programmes to be debugged should be a formal criteria for all systems developed within a structured programming environment. Where possible, fourth generation tools should be evaluated for their ability to permit the programmer to design debugging considerations into programmes. Programmes designed and written to be debugged are generally the simplest to debug.

Implemented within structured programming guidelines, but acceptable

within any standardized programming environment, this approach to programme development inherently leads to a definable and systematic approach to the resolution of any error. An informal, but equally recognizable aspect of the philosophy of designing programmes for debugging is that debugging should occur within a defined framework, providing both the system developer and the system user with a common understanding of how the task will be undertaken.

Once the task of debugging is placed within a defined structure, then the procedures and techniques for investigation may not only be described to the user community, but designed to be employed by the users themselves. Giving the user sufficient insight to the workings of computer processes that are executing tasks on their behalf reduces the possibility of perceived errors, and strengthens the understanding between the system user and the system developer at all stages of system development. Increasing the direct involvement of the user in the debugging process also reduces the further potential for misunderstanding during the initial stage of debugging that involves identifying the problem and describing it in terms that are common to both the user and the programmer.

Philosophically, and often for practical reasons, the task of debugging should begin with the programme input and output that the user perceives to be in error. Re-compilation of source code should not be necessary until the cause of the problem is clear, and the resolution has been written into the programme. Meeting this criteria demands that the implemented programme include designed logic to facilitate the gathering of all information necessary to isolate any process of the programme that may be in error. Preferrably, this logic should be conditional upon parameters passed during the execution of the programme.

The composition of the above elements presents one philosophy on debugging computer programmes that places emphasis on formalizing debugging considerations in the design of computer systems. Unfortunately, this does not directly address the potential for design errors themselves, but, since the system developer/system user relationship is enhanced through more direct user involvement in the debugging process, the potential is reduced indirectly. The introduction of debugging in the design of computer systems also improves the understanding amoung all concerned parties of the intent and use of the systems or programmes implemented.

Some Tools and Techniques.

The philosophical approach to debugging described above postulates that debugging considerations are a critical element of programme design. This section of the paper presents, by way of example, some tools and techniques that help to implement a design based approach to debugging that also reduces the need to recompile source code during the investigatory stage of debugging.

The example that follows is presented within the context of a COBOL programme with some called routines written in SPL. Many, if not all of the techniques illustrated may be implemented with other programming languages.

The discussion will evolve around the following programme as participants develop a given case study that will be included with the presentation materials. The following tools and techniques will be illustrated during the discussion to give the participants some insight to their potential uses.

MPE DEBUG,
Debugging mode in Cobol,
Job Control Words,
Copy Libraries,
Segmentation, and
Process timing techniques.

```
$Control  nolist,   source,   warn,   map,   code&
$     ,   bounds,   crossref,   locking,   mixed&
$     ,   verbs, quote=', uslinit
$Set X9=off
*     Off=Test Compilation, On=Implementation Compilation.
$Set X0=on
*     Off=SPL, On=Cobol.
$IF X9=off
$Control DEBUG
$IF
 Identification Division.
 Program-id.
      Pgm-Manager.
 Remarks.
      Summary

      Usage

      Installation

      Operation
           RUN pgm; parm=
                0, or not defined is normal operation.
                1, is 'process-debug' (process-activity-debug).
                64, is 'not UPDATING' (UPDATE-SWITCH).
           Input
           Processing
           Output
           Recovery/Debugging

 Environment Division.
 Copy A20b200 of A20bLIB nolist.
 Data Division.
 File Section.
 Working-Storage Section.
 Copy A20b300 of A20bLIB nolist
      replacing ==='$Title                                    '== by
                ==='$Actual Title                             '==.
 Procedure Division.
 Copy A20b600 of A20bLIB nolist.
 Function-Process.
      Display 'Virgin programme, no functions defined.'
           upon sysout.
*Include functional procedure files.
 Copy A20b800 of A20bLIB nolist.
$Control    list
*Segmentation.
*     Capability IA, PH are required.
$Control  nolist
 End-of-programme Section 99.
```

```
Configuration Section.                                          A20B200
Source-computer.  HP300048 with debugging mode.                 A20B200
Object-computer.  HP3000xx.                                     A20B200
Special-names.                                                  A20B200
     CONDITION-CODE is istatus,                                 A20B200
     SW9 is UPDATE-SWITCH, off status is UPDATING,              A20B200
     TOP is PAGE-EJECT.                                         A20B200
```

```
*Programme.                                                    A20B300
 01 Programme pic x(28) value spaces.                          A20B300
 01 Father-pin pic s9(4) comp value 0.                         A20B300
    88 father-process value 0.                                 A20B300
 01 process-control.                                           A20B300
    11 process-activity pic x(3) value spaces.                 A20B300
       88 process-end value 'END'.                             A20B300
    11 process-activity-debug pic x(1) value spaces.           A20B300
       88 process-debug value 'l'.                             A20B300
 01 flag-dialogue pic s9(4) comp value 0.                      A20B300
    88 process-dialogue value 1.                               A20B300
 01 process-error pic s9(4) comp value 0.                      A20B300
    88 no-process-error value 0.                               A20B300
*Programme Title.                                              A20B300
 01 Programme-Name.                                            A20B300
    11 Title pic x(40) value                                   A20B300
       '$Title                                  '.             A20B300
    11 filler pic x(26) value spaces.                          A20B300
    11 programme-date pic x(8) value spaces.                   A20B300
    11 filler pic x(1) value spaces.                           A20B300
    11 programme-time pic x(5) value spaces.                   A20B300
                                                               A20B300
*Processing Variables.                                         A20B300
 01 fstatus pic x(2) value spaces.                             A20B300
    88 fstatus-ok value '00'.                                  A20B300
 01 lgth pic s9(4) comp value 0.                               A20B300
 01 mpe-error pic s9(4) comp value 0.                          A20B300
 01 number-out pic 9(5)- value zeroes.                         A20B300
 01 numchar pic s9(4) comp value 0.                            A20B300
 01 passed-info pic x(80) value spaces.                        A20B300
 01 passed-parm pic s9(4) comp value 0.                        A20B300
 01 pin pic s9(4) comp value 0.                                A20B300
 01 work.                                                      A20B300
    11 work-1 pic s9(9) comp value 0.                          A20B300
    11 work-2 pic s9(9) comp value 0.                          A20B300
    11 work-3 pic s9(9) comp value 0.                          A20B300
*CreateProcess Variables.                                      A20B300
 01 process-items.                                             A20B300
    11 items pic s9(4) comp occurs 13 times.                   A20B300
 01 process-itemvalues.                                        A20B300
    11 itemvalue-1 pic x(2) value spaces.                      A20B300
    11 itemvalue-2 pic x(2) value spaces.                      A20B300
    11 itemvalue-3 pic x(2) value spaces.                      A20B300
    11 itemvalue-4 pic x(2) value spaces.                      A20B300
    11 itemvalue-5 pic x(2) value spaces.                      A20B300
    11 itemvalue-6 pic x(2) value spaces.                      A20B300
    11 itemvalue-7 pic x(2) value spaces.                      A20B300
    11 itemvalue-8 pic x(2) value spaces.                      A20B300
    11 itemvalue-9 pic x(2) value spaces.                      A20B300
    11 itemvalue-10 pic x(2) value spaces.                     A20B300
    11 itemvalue-11 pic x(2) value spaces.                     A20B300
    11 itemvalue-12 pic x(2) value spaces.                     A20B300
    11 itemvalue-13 pic x(2) value spaces.                     A20B300
```

```
*Date Variables.                                                A20B300
 01 work-dates.                                                 A20B300
    11 work-date-day pic s9(4) comp value 0.                    A20B300
    11 work-date-julian pic s9(9) comp value 0.                 A20B300
    11 work-date-yyyymmdd.                                      A20B300
       21 work-date-century pic x(2) value spaces.              A20B300
       21 work-date-yymmdd.                                     A20B300
          31 work-date-yy pic 9(2) value zeroes.                A20B300
          31 work-date-mm pic 9(2) value zeroes.                A20B300
          31 work-date-dd pic 9(2) value zeroes.                A20B300
*Timer Variables.                                               A20B300
 01 timer-1 pic s9(9) comp value 0.                             A20B300
 01 timer-2 pic s9(9) comp value 0.                             A20B300
 01 timer-debug-1 pic s9(9) comp value 0.                       A20B300
 01 timer-debug-2 pic s9(9) comp value 0.                       A20B300
 01 timer-out pic s9(9) sign leading separate.                  A20B300
 01 work-time.                                                  A20B300
    11 work-time-hh pic 9(2) value zeroes.                      A20B300
    11 work-time-mm pic 9(2) value zeroes.                      A20B300
    11 work-time-ss pic 9(2) value zeroes.                      A20B300
                                                                A20B300
*Input.                                                         A20B300
 01 stdinx pic s9(4) comp value 0.                              A20B300
 01 function-input.                                             A20B300
    11 function-type pic x(1) value spaces.                     A20B300
    11 function-command pic x(79) value spaces.                 A20B300
                                                                A20B300
*Output.                                                        A20B300
 01 stdlist pic s9(4) comp value 0.                             A20B300
 01 clear-screen.                                               A20B300
    11 filler pic x(1) value %33.                               A20B300
    11 filler pic x(1) value 'h'.                               A20B300
    11 filler pic x(1) value %33.                               A20B300
    11 filler pic x(1) value 'J'.                               A20B300
 01 cr pic x(1) value %15.                                      A20B300
 01 end-continue.                                               A20B300
    11 filler pic x(1) value %15.                               A20B300
    11 filler pic x(1) value %12.                               A20B300
    11 filler pic x(38) value                                   A20B300
       '     Please hit "Return" to continue.;'.                A20B300
 01 error-messages.                                             A20B300
    11 filler pic x(50) value                                   A20B300
       'Error, can not open $STDINX, or $STDLIST          '.    A20B300
    11 filler pic x(50) value                                   A20B300
       'Error, JCW or CIERROR not zero                    '.    A20B300
    11 filler pic x(50) value                                   A20B300
       'Error, programme jcw not zero                     '.    A20B300
    11 filler pic x(50) value                                   A20B300
       'Error, software license is not valid              '.    A20B300
 01 errormessages redefines error-messages.                     A20B300
    11 error-message pic x(50) occurs 4 times.                  A20B300
 01 error-limit pic s9(4) comp value 4.                         A20B300
 01 format-off.                                                 A20B300
    11 filler pic x(1) value %33.                               A20B300
    11 filler pic x(1) value 'X'.                               A20B300
```

```
01 lf pic x(1) value %12.                                    A20B300
01 message-buffer pic x(160) value spaces.                   A20B300
01 prompt-function.                                          A20B300
   11 filler pic x(1) value %33.                             A20B300
   11 filler pic x(7) value '&a2r00C'.                       A20B300
   11 filler pic x(9) value 'Function?'.                     A20B300
```

```
Declarative.                                                    A20B600
Debugger Section 01.  Use for debugging on all procedures.      A20B600
Debug-Dump.                                                     A20B600
     Display                                                    A20B600
          '                    ',                               A20B600
          '* * * * *  D e b u g  D u m p  * * * * *'.           A20B600
     Display                                                    A20B600
          '                    ',                               A20B600
          '          ========================          '.       A20B600
     Display DEBUG-ITEM, cr, lf.                                A20B600
Debug-Timer.                                                    A20B600
     Call intrinsic "PROCTIME" giving timer-debug-1.            A20B600
     IF timer-debug-2 = 0 then                                  A20B600
          move '1' to process-activity-debug                    A20B600
          call intrinsic "GETINFO"                              A20B600
               using passed-info, \\, passed-parm               A20B600
               giving mpe-error                                 A20B600
          move passed-parm to number-out                        A20B600
          display                                               A20B600
               'Information Passed', cr, lf, passed-info        A20B600
               , 'Parm = ', number-out                          A20B600
     ELSE                                                       A20B600
          compute timer-out = timer-debug-1 - timer-debug-2     A20B600
          display                                               A20B600
               '     '                                          A20B600
               , CURRENT-DATE                                   A20B600
               , ' '                                            A20B600
               , TIME-OF-DAY                                    A20B600
               , ' CPU time (milliseconds) since last point '   A20B600
               , timer-out.                                     A20B600
     Compute timer-debug-2 = timer-debug-1.                     A20B600
Debug-Dump-End.                                                 A20B600
     Display                                                    A20B600
          '=========================================='         A20B600
          , '==========================================='.      A20B600
     IF process-dialogue then                                   A20B600
          Display 'Debug end - please hit "Return" to continue.' A20B600
          Accept pin                                            A20B600
          Compute pin = 0.                                      A20B600
End Declaratives.                                               A20B600
```

```
Pgm-Main Section 11.                                                    A20B600
Perform Initializations.                                                A20B600
IF no-process-error then perform Processes until process-end            A20B600
ELSE next sentence.                                                     A20B600
Perform Conclusions.                                                    A20B600
GOBACK.                                                                 A20B600
                                                                        A20B600
Initializations.                                                        A20B600
     Move CURRENT-DATE to programme-date.                               A20B600
     Move TIME-OF-DAY to work-time.                                     A20B600
     String work-time-hh delimited by size                              A20B600
          , ":" delimited by size                                       A20B600
          , work-time-mm delimited by size                              A20B600
          into programme-time                                           A20B600
     Call intrinsic "PROCINFO"                                          A20B600
          using lgth, numchar, \0\, \10\, programme.                    A20B600
     Call intrinsic "FOPEN" using \\, \%254\ giving stdinx.             A20B600
     IF istatus = 0 then next sentence ELSE compute stdinx = 0.         A20B600
     Call intrinsic "FOPEN"                                             A20B600
          using \\, \%214\, \%1\ giving stdlist.                        A20B600
     IF istatus = 0 then next sentence ELSE compute stdlist = 0.        A20B600
     IF stdinx = 0 OR stdlist = 0 then compute process-error = 1.       A20B600
     String format-off delimited by size                                A20B600
          , clear-screen delimited by size                              A20B600
          , programme-name delimited by size                            A20B600
          into message-buffer.                                          A20B600
     Call intrinsic "PRINT" using message-buffer, \-86\, \%40\.         A20B600
     IF no-process-error then                                           A20B600
          call intrinsic "GETJCW" giving mpe-error                      A20B600
          IF mpe-error = 0 then                                         A20B600
               move 'CIERROR.' to message-buffer                        A20B600
               call intrinsic "FINDJCW"                                 A20B600
                    using message-buffer, pin, mpe-error                A20B600
               IF mpe-error = 0 and pin = 0 then                        A20B600
                    call intrinsic "FINDJCW"                            A20B600
                         using programme, pin, mpe-error                A20B600
                    IF mpe-error = 3                                    A20B600
                    OR (mpe-error = 0 and pin = 0) then                 A20B600
                         call "PROCESS'PROFILE"                         A20B600
                              using \stdinx\, flag-dialogue             A20B600
                         call intrinsic "FATHER" using father-pinA20B600
                         IF istatus = 0 then NEXT SENTENCE              A20B600
                         ELSE compute father-pin = 0                    A20B600
                    ELSE compute process-error = 3                      A20B600
               ELSE compute process-error = 2                           A20B600
          ELSE compute process-error = 2                                A20B600
     ELSE next sentence.                                                A20B600
```

```
Conclusions.                                                    A20B600
     Move CURRENT-DATE to programme-date.                       A20B600
     Move TIME-OF-DAY to work-time.                             A20B600
     String work-time-hh delimited by size                      A20B600
          , ":" delimited by size                               A20B600
          , work-time-mm delimited by size                      A20B600
          into programme-time.                                  A20B600
     Call intrinsic "PUTJCW"                                    A20B600
          using programme, process-error, mpe-error.            A20B600
     IF no-process-error then                                   A20B600
          display                                               A20B600
               cr, lf, programme-date, ' ', programme-time, '.  'A20B600
               , 'Normal end of programme.' UPON SYSOUT         A20B600
     ELSE                                                       A20B600
          move process-error to number-out                      A20B600
          string "Programme Error " delimited by size           A20B600
               , number-out delimited by size                   A20B600
               , "." delimited by size                          A20B600
               into message-buffer                              A20B600
          call intrinsic "PRINT"                                A20B600
               using message-buffer, \-22\, \%40\               A20B600
          IF process-error < 0                                  A20B600
          OR process-error > error-limit then                   A20B600
               Display                                          A20B600
                    cr, lf, programme-date                      A20B600
                    , ' ', programme-time, '.  '                A20B600
                    , 'Abnormal end of programme.' UPON SYSOUT  A20B600
               compute process-error = %100000                  A20B600
               call intrinsic "SETJCW" using process-error      A20B600
          ELSE                                                  A20B600
               Display                                          A20B600
                    cr, lf, error-message(process-error)        A20B600
                    , cr, lf, programme-date                    A20B600
                    , ' ', programme-time, '.  '                A20B600
                    , 'Abnormal end of programme.' UPON SYSOUT. A20B600
```

```
Processes.                                                          A20B600
     Move spaces to function-input.                                 A20B600
     Move CURRENT-DATE to programme-date.                           A20B600
     Move TIME-OF-DAY to work-time.                                 A20B600
     String work-time-hh delimited by size                          A20B600
          , ":" delimited by size                                   A20B600
          , work-time-mm delimited by size                          A20B600
          into programme-time                                       A20B600
     String format-off delimited by size                            A20B600
          , clear-screen delimited by size                          A20B600
          , programme-name delimited by size                        A20B600
          , prompt-function delimited by size                       A20B600
          into message-buffer.                                      A20B600
     Call "INPUT" using message-buffer, function-input, \80\.       A20B600
     IF function-input = spaces then                                A20B600
          call intrinsic "FATHER" giving father-pin                 A20B600
          IF istatus = 0 then                                       A20B600
               call intrinsic "ACTIVATE" using \0\, \3\             A20B600
          ELSE move 'END' to process-control                        A20B600
     ELSE                                                           A20B600
          call "EDIT'UP" using function-input, \80\                 A20B600
          IF function-input = 'EXIT' OR 'END' then                  A20B600
               move 'END' to process-control                        A20B600
          ELSE                                                      A20B600
               IF function-type = ':' then                          A20B600
                    perform Command                                 A20B600
                    call "INPUT'0" using end-continue, pin, \1\     A20B600
               ELSE                                                 A20B600
                    IF function-type = '$' then                     A20B600
                         perform CreateProcess                      A20B600
                         call "INPUT'0"                             A20B600
                              using end-continue, pin, \1\          A20B600
                    ELSE                                            A20B600
                         perform Function-Process                   A20B600
                         IF no-process-error then                   A20B600
                              call "INPUT'0"                        A20B600
                              using end-continue, pin, \1\          A20B600
                         ELSE                                       A20B600
                              move process-error to number-out      A20B600
                              compute process-error = 0             A20B600
                              string                                A20B600
                                   "     Process Error "            A20B600
                                        delimited by size           A20B600
                                   , number-out                     A20B600
                                        delimited by size           A20B600
                                   , "." delimited by size          A20B600
                                   into message-buffer              A20B600
                              call intrinsic "PRINT"                A20B600
                                   using message-buffer             A20B600
                                        , \-25\, \%40\              A20B600
                              call "INPUT'0"                        A20B600
                                   using end-continue, pin, \1\.    A20B600
```

```
Command.                                                        A20B600
     Move spaces to message-buffer.                             A20B600
     String function-command delimited by size                  A20B600
          , cr delimited by size                                A20B600
          into message-buffer.                                  A20B600
     Call intrinsic "COMMAND" using                             A20B600
          message-buffer, mpe-error, pin.                       A20B600
     IF mpe-error = 0 then NEXT SENTENCE                        A20B600
     ELSE                                                       A20B600
          move mpe-error to number-out                          A20B600
          IF mpe-error < 0 then                                 A20B600
               string "Command Warning " delimited by size      A20B600
                    , number-out delimited by size              A20B600
                    into message-buffer                         A20B600
               call intrinsic "PRINT"                           A20B600
                    using message-buffer, \-21\, \%40\          A20B600
          ELSE                                                  A20B600
               string "Command Error " delimited by size        A20B600
                    , number-out delimited by size              A20B600
                    into message-buffer                         A20B600
               call intrinsic "PRINT"                           A20B600
                    using message-buffer, \-19\, \%40\.         A20B600
     Move spaces to message-buffer, function-input.             A20B600
     Compute mpe-error = 0.                                     A20B600
     Compute pin = 0.                                           A20B600
```

```
CreateProcess.                                                  A20B600
     Move spaces to message-buffer.                             A20B600
     Move function-command to message-buffer.                   A20B600
     Compute items(1) = 10.                                     A20B600
     Compute items(2) = 6.                                      A20B600
     Compute items(3) = 3.                                      A20B600
     Compute items(4) = 0.                                      A20B600
     Move %3 to itemvalue-1.                                    A20B600
     Move %75000 to itemvalue-2.                                A20B600
     Move %41 to itemvalue-3.                                   A20B600
     Move 0 to itemvalue-4.                                     A20B600
     Call intrinsic "CREATEPROCESS"                             A20B600
          using mpe-error, pin, message-buffer                  A20B600
          , process-items, process-itemvalues.                  A20B600
     IF mpe-error = 0 then NEXT SENTENCE                        A20B600
     ELSE                                                       A20B600
          move mpe-error to number-out                          A20B600
          IF mpe-error < 0 then                                 A20B600
               string "Loader Warning " delimited by size       A20B600
                    , number-out delimited by size              A20B600
                    into message-buffer                         A20B600
               call intrinsic "PRINT"                           A20B600
                    using message-buffer, \-20\, \40\           A20B600
          ELSE                                                  A20B600
               string "Loader Error " delimited by size         A20B600
                    , number-out delimited by size              A20B600
                    into message-buffer                         A20B600
               call intrinsic "PRINT"                           A20B600
                    using message-buffer, \-18\, \%40\.         A20B600
     Call intrinsic "KILL" using pin.                           A20B600
     Move spaces to message-buffer, function-input.             A20B600
     Compute mpe-error = 0.                                     A20B600
     Compute pin = 0.                                           A20B600
```

```
Subroutines Section 91.                                          A20B800
SUBROUTINE-Start-Clock.                                          A20B800
     Call intrinsic "PROCTIME" giving timer-1.                   A20B800
     Display                                                     A20B800
          "Timer     "                                           A20B800
          , CURRENT-DATE                                         A20B800
          , " "                                                  A20B800
          , TIME-OF-DAY                                          A20B800
          , " Clock Started.".                                   A20B800
                                                                 A20B800
SUBROUTINE-Stop-Clock.                                           A20B800
     Call intrinsic "PROCTIME" giving timer-2.                   A20B800
     Compute timer-out = timer-1 - timer-2.                      A20B800
     Display                                                     A20B800
          "Timer     "                                           A20B800
          , CURRENT-DATE                                         A20B800
          , " "                                                  A20B800
          , TIME-OF-DAY                                          A20B800
          , " Clock Stopped: CPU used (milliseconds) "           A20B800
          , timer-out.                                           A20B800
```

```
Procedure Process'Profile(stdinx,flag);
     Value   stdinx;
     Integer stdinx, flag;
!This procedure defines whether programmatic dialogue may occur.
BEGIN
INTRINSIC FFILEINFO, WHO, PRINTFILEINFO;
Logical array  mode(0:0);
Byte array     mode'ba(*)=mode;

!Input the user's profile.
WHO(mode);
IF mode.(12:2)=%1<<session>> then BEGIN
     FFILEINFO(stdinx,2,mode'ba);
!If it is recognized that the user is operating a session, and that
!stdinx (opened in Initializations) is either $STDIN or $STDINX, then
!the programme may expect the user to respond to prompting.  Otherwise,
!the user is passing input into the programme using a file, and it is
!logically incorrect to expect the user to respond to alternate action
!prompting (from a recoverable exception).
     IF %4<=INTEGER(mode.(10:3))<=%5 then flag:=1
     ELSE flag:=0;
END
ELSE flag:=0;
END<<Process'Profile>>;
```

```
Procedure Input(parameter0,parameter1,parameter2);
     Value parameter2; Integer parameter2;
     Array parameter0<<0:79>>, parameter1;
<<
Summary       : This procedure outputs the prompt, ending with "?", in
                parameter 0, and returns the input user response in
                parameter 1.  The length of the expected response is
                passed in parameter 2.
>>
BEGIN
INTRINSIC PRINT, READX;

Byte array      input'ba(*)=parameter1;
Array           message(0:79);
     Byte array      message'ba(*)=message;
Byte array      prompt(*)=parameter0;
Integer         lgth:=0;

lgth:=SCAN prompt UNTIL "?";
message:="  ";
MOVE message(1):=message,(79);
MOVE message'ba:=prompt,(lgth+1);
PRINT(message,-(lgth+2),%320);
input'ba:=" ";
MOVE input'ba(1):=input'ba,(parameter2);
lgth:=READX(parameter1,-(parameter2+1));
END<<Input>>;

Procedure Input'0(parameter0,parameter1,parameter2);
     Value parameter2; Integer parameter2;
     Array parameter0<<0:79>>, parameter1;
<<
Summary       : This procedure outputs the prompt, delimited by ";",
                in parameter 0, and returns the input user response in
                parameter 1.  The expected length of the user response
                is passed in parameter 2.
>>
BEGIN
INTRINSIC PRINT, READX;

Byte array      input'ba(*)=parameter1;
Array           message(0:79);
     Byte array      message'ba(*)=message;
Byte array      prompt(*)=parameter0;
Integer         lgth:=0;

lgth:=SCAN prompt UNTIL ";";
message:="  ";
MOVE message(1):=message,(79);
MOVE message'ba:=prompt,(lgth);
PRINT(message,-(lgth+1),%320);
input'ba:=" ";
MOVE input'ba(1):=input'ba,(parameter2);
lgth:=READX(parameter1,-(parameter2+1));
END<<Input'0>>;
```

```
Procedure Edit'Up(parameter0,parameter1);
     Value parameter1; Integer parameter1;
     Array parameter0;
<<
Summary       : This procedure edits a string of characters passed in
                parameter 0 for a length of characters defined in
                parameter 1.  Any lower case letters are shifted up.
>>
BEGIN
Byte array buffer'ba(*)=parameter0;
parameter1:=parameter1 - 1;
DO BEGIN
     IF %141<=INTEGER(buffer'ba(parameter1))<=%172 then
          buffer'ba(parameter1):=buffer'ba(parameter1) - %40;
     parameter1:=parameter1 - 1;
END UNTIL parameter1 < 0;
END<<Edit'Up>>;
```