UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

# COMPUTER CENTRE BULLETIN

*Volume 3, Number 8.*
*3rd August, 1970.*

*Editor:*
*S. J. Barry*

## THIS EDITION

The editor apologizes for any delay in this month's Bulletin, caused by her recent absence. The main article in this month's Bulletin is on the release of PDP-10 absolute overlays. This will be available to all users of the PDP-10.

## STAFF NEWS

We are sorry to say goodbye to two members of the Computer Centre Staff; *Mr. John Row* and *Mrs. Diann Munro*. John joined the staff in January 1969, having previously obtained his Electrical Engineering degree and Diploma in Automatic Computing at this University. He has moved to Sydney to become a Systems Programmer with Control Data Corporation.

Diann, a data preparation assistant, has also left to take up a new position.

We wish both John and Diann every success in their new work.

## GE 225 FORTRAN IV

It is necessary to remind all users of the GE 225 that the console switches are not available to them. Any programs attempting to use these switches cannot be accepted by the Centre.

A modification to the absolute suffix is shortly to be introduced to improve the card-reader error handling procedures. This change should prevent the apparent double reading of data cards due to card reader misfeeds.

# PDP-10 FORTRAN IV

## 1. COMPILER ERRORS

(a) It has been found that an expression of the form -

```
A(1,J) = A(N,J)
```

does not compile properly. Until D.E.C. effect a change to the compiler, this can be overcome by replacing the expression with something of the form -

```
XXX = A(N,J)
A(1,J) = XXX
```

This error appears when similar subscript calculations occur within an expression, with the second subscript being a variable and the first alternately a variable and a constant.

(b) In attempting to optimise the use of registers when evaluating expressions, the compiler occasionally produces incorrect code. The major area of difficulty concerns the use of common negated sub-expressions, of which the example shown is typical. (A*B is the common sub-expression.)

```
PT1 = (A*B - B*C) *FA + (C*A - A*B) *FB
```

## 2. CONSTANTS IN SUBROUTINE CALLS

Assignment statements for a constant within a subroutine call will alter that constant. Further references to the constant will therefore give an incorrect answer.

example:

```
CALL Q(1.Ø,65ØØØ.1)
A=1.Ø
B=65ØØØ.1
WRITE(5,1) A,B
FORMAT(2F16.2)
END
SUBROUTINE Q(X,Y)
Y=X
RETURN
END
```

will give the results

```
1.ØØ            1.ØØ
```

The reason for this is because the compiler processes the source program in only one pass.

If an actual argument corresponds to a dummy argument that is defined or redefined in the referenced subprogram, the actual argument must be a variable; a name, an array element name or array name.

When the compiler is compiling a subroutine it has no way of knowing if the calling program will violate the above rule.

## 3. PRINTING OF DOUBLE PRECISION VALUES

A modification has been made to the re-entrant FORTRAN operating system to correct an error when printing double precision variables using the 'D' format. The modification has improved the situation but, as yet, not fully corrected it.

Values in the range $\emptyset.1 \times 1\emptyset^{-16}$ to $\emptyset.1 \times 1\emptyset^{+8}$ are now printed correctly. Outside this range, results are inconsistent and definitely not reliable.

## PDP-10 SYSTEM CHANGES

### 1. NEW RELEASE

On Monday 3rd August, a new version of the Batch processing control program is to be released for general use. Some new facilities have been introduced, but the main purpose of the release is to improve efficiency and overcome errors previously reported. While every attempt has been made to ensure that errors have been eliminated, it is possible that some errors have not been corrected or further errors may have been introduced. The complexity of the major elements of the system is such that it is not possible to test for all possible error conditions and the Computer Centre cannot be responsible for the effects of malcalculation inadvertently introduced by this or other system changes. We would ask your help in reporting suspected errors promptly, so that they may be investigated and corrected as soon as possible.

It is the user's responsibility to include some degree of consistency checking into his programs to guard against the possibility of incorrect results caused by hardware or software errors. The user should also be aware that discrepancies may occur, and particularly after a system change should check runs carefully for discrepancies.

### 2. NEW FACILITIES

The major new facility provided in the release is the availability of a program overlay capability. Details of this facility are given in the following article entitled "PDP-10 Absolute Overlays".

## 3. OPTIONS

A further option is now available to the .FORTRAN control card, - NOBIN - which inhibits the output of the binary file normally produced by a compilation. Each .FORTRAN command normally produces a binary file, whose name is kept on an internal list, and on the occurrence of a .RUN command, if there are no errors, all the binary files on the list are loaded and brought into execution. The occurrence of the .RUN command also zeros the list, so a subsequent .RUN will not cause the loading of spurious files. The NOBIN option prevents the creation of the binary file and should facilitate the interleaving of debug compilations with compile and run sequences.

## 4. IMPROVEMENTS

An attempt has been made in this version to remedy known errors in the batch system, particularly those which caused suppression of some error messages and suppression of the listing if a CREF option had been selected on the .FORTRAN command and compilation errors had occurred. The SYMBOL option in the .RUN card will now correctly produce a full SYMBOL table after errors occur during program execution. This option is an extremely valuable aid in the debugging of programs.

An attempt has been made to improve the throughput of the system, and this should result in improved job costs. The new loader which was released on 15th July should also result in a reduction of program loading costs.

## PDP-10 ABSOLUTE OVERLAYS

*R.D. Nilsson*

*The author of this article, Mr. Ron Nilsson, is Senior Lecturer in the Department of Civil Engineering. Mr. Nilsson has contributed a great deal towards the work of the Computer Centre and this overlay system for the PDP-10 has been designed to give at least the same facilities as are presently available on the GE 225.*

### 1. ABSTRACT

This memorandum describes the structure and usage of an absolute overlay system. Modifications have been made to version 51 of the Linking Loader (to create the required overlays) and to BATCH (to issue the required commands). A subroutine OVERLAY is provided in LIB40 which can be used from a FORTRAN program to call the overlays.

## 2. OVERLAY STRUCTURE

An overlaid program consists of a permanent segment and one or more overlaid segments. Each segment can contain one or more FORTRAN functions, but the MAIN program must be in the permanent segment.

The program overlay structure may conveniently be described in terms of a tree structure as shown in Figure 1. There are a number of alternative branches in the tree, each representing one combination of coexisting segments in core. Communication can be carried out directly along branches, but only indirectly between branches through common branch links. The root of the tree is the permanent segment at the bottom of core and the branches reach upwards towards the top of core. Each overlay segment or link has three attributes:

(a) Overlay Number - A decimal integer representing the order of the overlay at creation time, i.e., the order in which they are presented to the linking loader. This number may be used in calling subroutine OVERLAY to indicate which particular overlay is to be called. In Figure 1, the overlay numbers are circled.

(b) Overlay Area - An octal integer indicating the topological area of core into which the overlay segment is to be loaded. The overlay areas are numbered consecutively from 1 along any branch of the tree. Area 1 is the first overlay area above the permanent area. In Figure 1 and Figure 2 the overlay areas are placed in quotes. An example of possible core usage is shown in Figure 2 and illustrates that the actual position of an area depends on those below it in the branch. The total core area used depends on the length of the longest branch.

(c) Overlay Name - An optional name with a maximum of five ASCII characters which may be used as an alternative to the overlay number when calling the overlay. The name may be the same as that of a subroutine within the overlay.

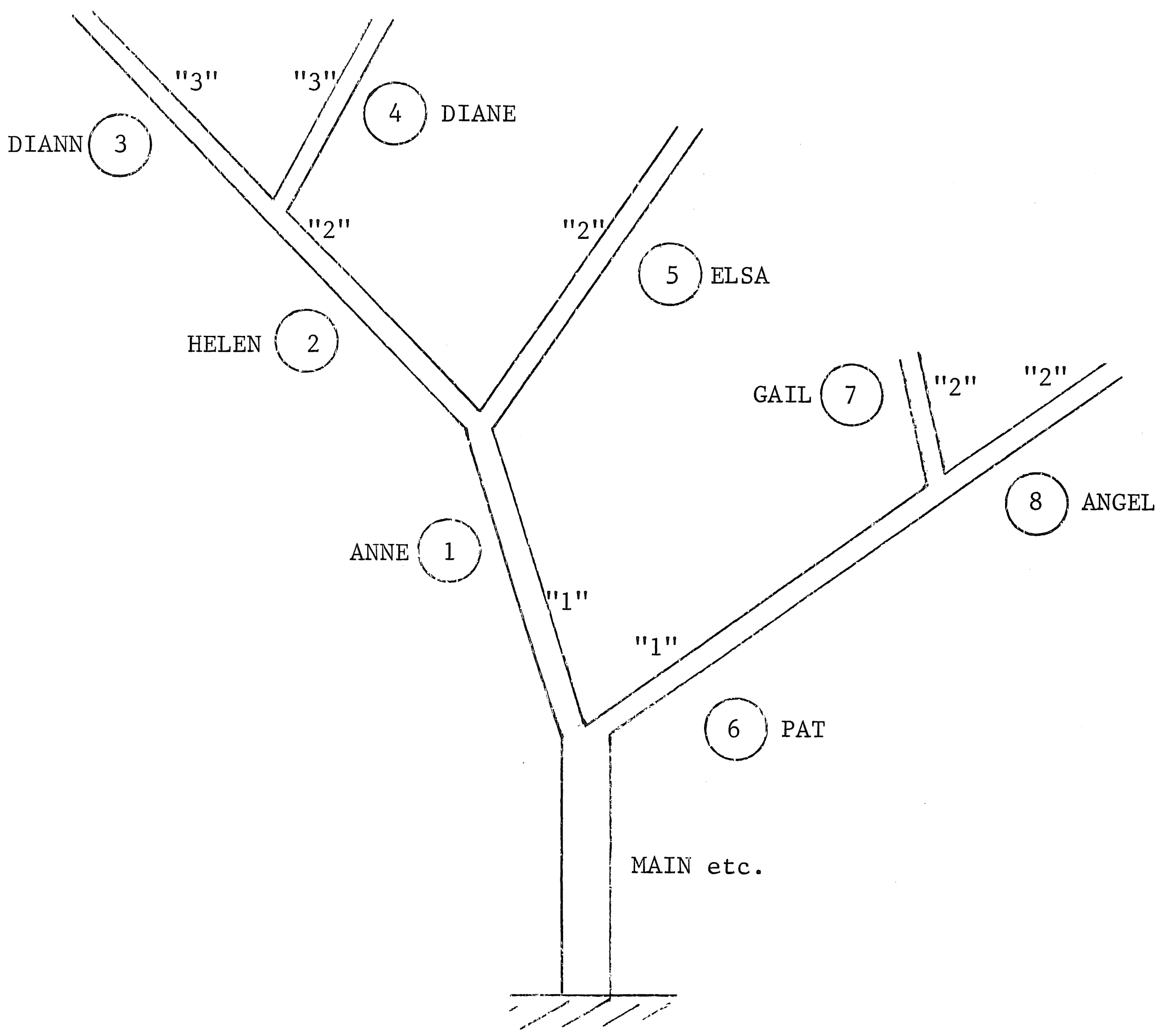

Figure 1.

| Number | Area | Name |
|---|---|---|
| 1 | 1 | ANNE |
| 2 | 2 | HELEN |
| 3 | 3 | DIANN |
| 4 | 3 | DIANE |
| 5 | 2 | ELSA |
| 6 | 1 | PAT |
| 7 | 2 | GAIL |
| 8 | 2 | ANGEL |

Table 1.

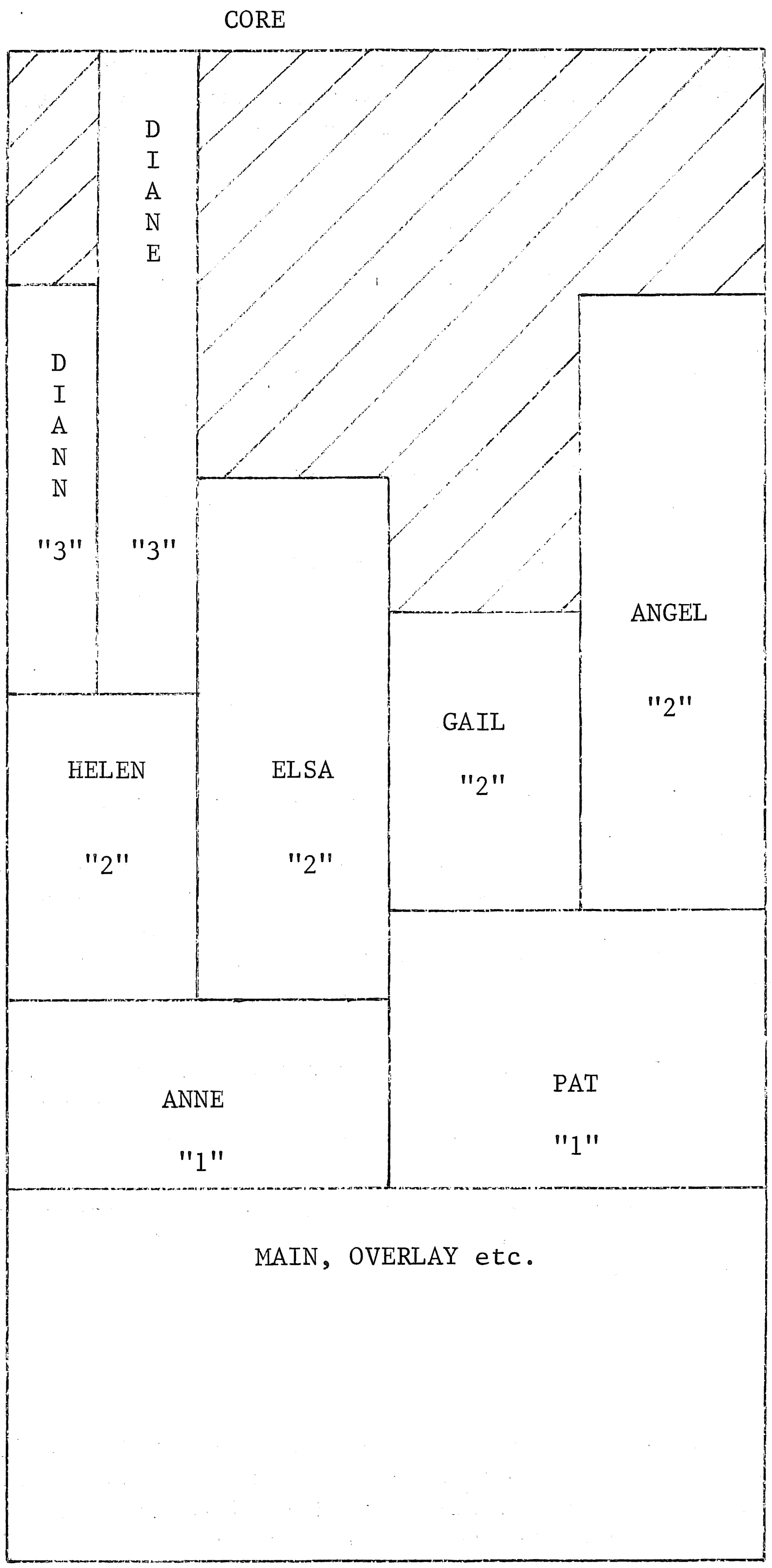
CORE
DIANE
DIANN
"3"
"3"
ANGEL
"2"
GAIL
"2"
HELEN
"2"
ELSA
"2"
ANNE
"1"
PAT
"1"
MAIN, OVERLAY etc.

Figure 2.

## 3. OVERLAY CREATION BY THE USER

The user must indicate where each overlay starts. This is done with an overlay separator which indicates the overlay area and its name. In BATCH, this would be done with the following sequence for the FORTRAN example given.

```
.FORTRAN
        Main segment routines

.OVERLAY  AREA=1, NAME=ANNE             ; OVERLAY 1

.FORTRAN
        Routines to be included in ANNE
.OVERLAY  2 HELEN                       ; OVERLAY 2
.FORTRAN
        Routines to be included in HELEN
.OVERLAY NAME=DIANN AREA=3              ; OVERLAY 3
.FORTRAN
        Routines to be included in DIANN
.OVERLAY 3, DIANE                       ; OVERLAY 4
.FORTRAN
        Routines to be included in DIANE
```

etc., following the order given in Table 1.

The .FORTRAN control cards can contain the normal options and there may be more than one .FORTRAN card in each overlay. However, each overlay must start with a control card immediately following the .OVERLAY card. Various ways of punching the .OVERLAY card are given, and it should be noted that the area must be given first if identifiers are not included. If no area is given, then an area of 1 is assumed. This allows a simple one area overlay program to be set up requiring only simple separators. The name is optional.

Note that it is the overlay area that is indicated on the card. The overlay number is the number of the overlay in the deck and this has been included in the example for clarity by putting it after a comment (;) indicator.

The order of the overlays in the deck is important as this determines the core layout. Each branch must be completed in an orderly fashion as described more fully in the following section.

## 4. OVERLAY CREATION BY THE SYSTEM

Each overlay separator creates a command to the linking loader indicating the overlay area and name to be used. On receipt of this command a library search is carried out so that any library routines mentioned in the last or old overlay, which do not already exist lower in the branch, can be included.

An entry is then created in an overlay table within the subroutine OVERLAY, containing the name and area information given. The overlay number is the number of the entry in the overlay table.

If the new overlay area number is greater than the old, the new overlay is loaded into core immediately above the old overlay.

However, one or more old overlays must be overwritten if the new area is less than or equal to the old area. In this case, the core images of these old overlays are written out on an overlay file and no further changes can be made to them. Information about size and position of these overlays is placed in the overlay table for later use by OVERLAY. The next location to be loaded is then reset so that the new overlay is placed in the area previously used by the old overlay(s).

For example, in Figures 1 and 2, consider the case when overlay ELSA is reached. The old overlay at this time is DIANE with an area 3 compared with ELSA's 2. Therefore DIANE is written out. Now the old overlay is HELEN with area 2 and this is again written out. Now the old overlay is ANNE with area 1 and ELSA can be added immediately above it.

Note that this writing out process is done during creation and it is at this stage that the overlay order is important as it determines the core usage and the possible interconnections between overlays. During actual execution of the program, the order of usage can be quite different and no writing out of an old overlay is done when a new overlay is brought in by OVERLAY. The areas into which the overlays are brought are fixed permanently at creation time.

An alternative way of numbering and presenting the overlays for the example is shown in Figure 3 and Table 2.

Alternative Representation of Figure 1.

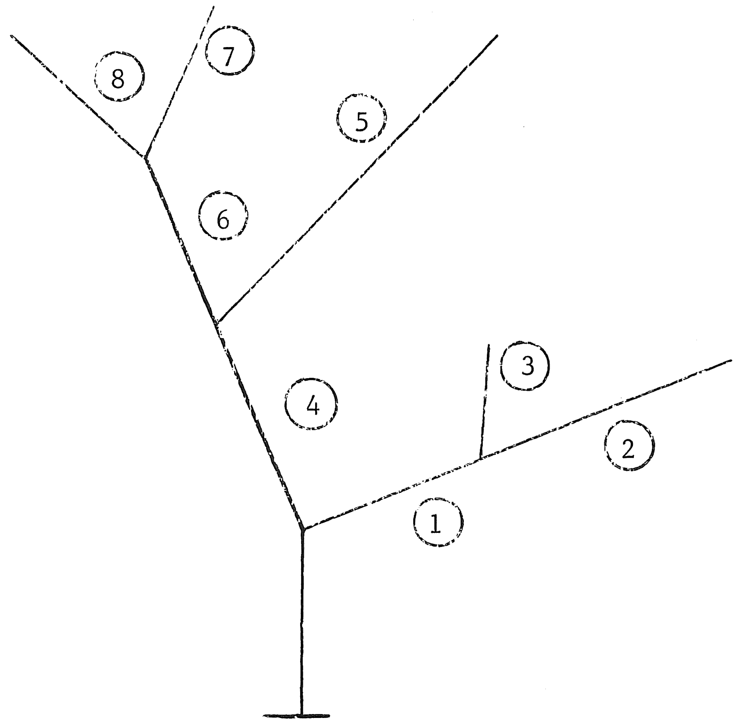

Figure 3.

| Number | Area | Name |
|---|---|---|
| 1 | 1 | PAT |
| 2 | 2 | ANGEL |
| 3 | 2 | GAIL |
| 4 | 1 | ANNE |
| 5 | 2 | ELSA |
| 6 | 2 | HELEN |
| 7 | 3 | DIANE |
| 8 | 3 | DIANN |

Table 2.

## 5. CALLING OVERLAYS

When the program is executed, no overlays are in core and an overlay must be brought into core before the routines in the overlay can be used. To call overlay 4 in the example, we could use one of the following statements:

```
(a)     CALL OVERLAY(4)
(b)     J=4
        CALL OVERLAY(J)
(c)     CALL OVERLAY ('DIANE')
(d)     NAME='DIANE'
        CALL OVERLAY(NAME)
```

DIANE will not be loaded if it is already in core.

Note that the call by name can only be used if the overlays have been named. If any overlay is unnamed its name entry in the overlay table will be zero.

When a call to load an overlay is made, overlays lower in the branch will also be loaded if they are not already in core. For example, HELEN and ANNE would be loaded if one of the above calls to DIANE were made from the main segment and if they were not already in core. Note that DIANN, ELSA, PAT, GAIL and ANGEL would be marked as not being in core.

When an overlay is loaded by a call to OVERLAY, its core image will be returned to its state at creation time. Therefore, values of variables set within the overlay during execution are lost when reloaded.

## 6. COMMUNICATION BETWEEN OVERLAYS

Because of the structure of the overlay system, there may be upwards and downwards communication between routines in a branch. Thus, routines in the main segment can call routines in any overlay and vice versa. In the example, routines in ANNE can call those in DIANE and vice versa, but those in DIANE cannot call those in ELSA. DIANE and ELSA can only pass data to each other via some common arguments from ANNE, or through a COMMON area located in ANNE or the main segment as described later.

## 7. LIBRARY ROUTINES

Since a library search is carried out at the end of each overlay and the main segment, library routines are placed in overlays as necessary. For example, if SINE were used by HELEN, it would be placed there if not already mentioned in ANNE or the main segment. DIANE and DIANN could both refer to HELEN's SINE. However, a second copy would be placed in ELSA if SINE was required there as well.

## 8. COMMON

Blank COMMON, named COMMON and BLOCK DATA can be thought of in the same way as library routines. They are placed in the first overlay in the branch that mentions them and are then available to overlays further up the branch. Thus, a COMMON defined in a main segment can be used by all overlays. However, a COMMON first defined in HELEN is available to HELEN, DIANN and DIANE only and will be reset to its original values every time HELEN is reloaded. Furthermore, if this COMMON is mentioned in ELSA, it will be an entirely different one to that provided in HELEN.

## 9. DATA STATEMENTS

DATA statements can give subtly different effects on reloading since they can refer to something in another area. It must be remembered that DATA statements are set up at creation time and not execution time. For example, suppose blank COMMON is defined in the main segment and suppose HELEN contains data statements that refer to blank COMMON. In this case, reloading of HELEN will not reset the data as they were set at creation time. However, if HELEN's data statements referred to variables in HELEN or to a COMMON that was first mentioned in HELEN, then they will be reset every time HELEN is reloaded as they have been placed in the core image overlay that is reloaded.

## 10. ERRORS DETECTED DURING CREATION

?MISSING OVERLAY SUBROUTINE
: There was no overlay call in the main segment

?OVERLAY TABLE FULL
: More than 2Ø overlays

?MISSING OVERLAY AREA
: Numbering of overlay areas incorrect

## 11. ERRORS DURING LOADING OF AN OVERLAY

All error messages generated by subroutine OVERLAY will indicate the name or number of the overlay which is being called and the octal address of the call. In addition, one of the following messages is output:

OVERLAY NUMBER INCORRECT
: A call to OVERLAY with a number Ø or greater than 2Ø. Note that a negative number is interpreted as a name.

OVERLAY NOT IN TABLE
: The name in the overlay call does not exist, perhaps the overlay was not named at creation time.

ERROR READING OVERLAY FILE

DISK UNAVAILABLE

OVERLAY FILE NOT PRESENT
Trouble whilst trying to read the overlay file

OVERLAY WILL OVERWRITE CALLER
For example, if DIANE tries to load ANNE or ELSA

## 12. <u>UNDETECTED ERRORS</u>

The user can cause errors that are undetected by the system and which will result in unpredictable results.

A call to a routine in an overlay that has not previously been loaded by a CALL OVERLAY or which has been overwritten by a later call, will obviously be incorrect.

A call to a routine in another branch can result if, for example, HELEN calls a routine that is incorrectly placed in ELSA. This leads to unpredictable results at creation time.

The following FORTRAN example illustrates the use of overlays.

```
.FORTRAN
      A1=Ø.Ø
      A2=Ø.Ø
      A3=Ø.Ø
C                                             READ RADIUS AND NUMBER
 5    READ 1Ø, A1, NUM
 1Ø   FORMAT  (F1Ø.4, I1)
      IF  (A1 .EQ. Ø.Ø)  GO TO 5Ø
C                                     NUM = 1 CIRCLE, 2 CYLINDER, 3 SPHERE
      IF  (NUM - 2)  1, 2, 3
C                                     CIRCLE
C                                             PERIMETER
C                                             AREA
 1    CALL OVERLAY ('CIRCL')
      CALL CIR (A1, A2, A3)
      PRINT 11
 11   FORMAT  (1H1/ 8H CIRCLE://35H      RADIUS   PERIMETER         AREA)
      GO TO 4Ø
C                                     CYLINDER
C                                             SURFACE AREA
C                                             VOLUME
 2    CALL OVERLAY ('CYLIN')
      CALL CIR (A1, A2, A3)
      CALL CYL (A1, A2, A3)
      PRINT 22
 22   FORMAT  (1H1/1ØH CYLINDER:/ 11H      HEIGHT/
     1        35H AND RADIUS  SURF. AREA      VOLUME)
      GO TO 4Ø
```

```
C                                     SPHERE
C                                             SURFACE AREA
C                                             VOLUME
 3    CALL OVERLAY ('SPHER')
      CALL SPH (A1, A2, A3)
      PRINT 33
 33   FORMAT  (1H1/ 8H SPHERE://35H     RADIUS  SURF. AREA      VOLUME)
      GO TO 4Ø
 4Ø   PRINT 45, A1, A2, A3
 45   FORMAT  (1H , F1Ø.4, 2F12.4)
      GO TO 5
 5Ø   END


.OVERLAY  AREA=1, NAME=CIRCL

.FORTRAN
      SUBROUTINE CIR (RAD, PER, ARE)
      PER = 2.Ø*3.14159*RAD
      ARE = 3.14159*RAD*RAD
      END


.OVERLAY  2,  CYLIN

.FORTRAN
      SUBROUTINE CYL (HGT, SAR, VOL)
      SAR = SAR*(HGT+HGT)
      VOL = VOL*HGT
      END


.OVERLAY  NAME=SPHER   AREA=1

.FORTRAN
      SUBROUTINE SPH (RAD, SAR, VOL)
      SAR = 4.Ø*3.14159*RAD*RAD
      VOL = (4.Ø*3.14159*RAD**3)/3.Ø
      END
```

When this program is run with the following data cards

```
|col. 1
|
|bbbb5.Øbbb1
|bbbb5.Øbbb2
|bbbb5.Øbbb3
|bbbbØ.Ø
```

the results are as follows -

```
CIRCLE:

    RADIUS   PERIMETER        AREA
    5.ØØØØ     31.4159     78.5397


CYLINDER:
    HEIGHT
AND RADIUS  SURF. AREA      VOLUME
    5.ØØØØ    314.159Ø    392.6987


SPHERE:

    RADIUS  SURF. AREA      VOLUME
    5.ØØØØ    314.159Ø    523.5983
```

## LIBRARY ACCESSIONS

This is a list of books acquired by the Libraries of the University of Queensland in May which may be of interest to readers of the Bulletin.

| | |
|---|---|
| Minsky, Marvin Lee, ed. | *Semantic Information Processing*. 1968. (001.535 MIN Engin.) |
| Rivett, Patrick. | *Concepts of operational research*. 1968. (001.424 RIV Math.) |
| Meadow, Charles T. | *An analysis of information systems*. 1967. (029.7 MEA Acc.) |
| Robbin, Joel W. | *Mathematical logic*. 1969. (164 ROB Main) |
| Diebold, John. | *Man and the computer*. 1969. (301.24 DIE Main) |
| *Data Processing for education*. v.9, 1970, and onwards. (370.184 DAT Cent. Med.) | |
| Carnahan, Brice. | *Applied numerical methods*. (Qto 517.6 CAR Engin.) |
| Hohn, Franz Edward. | *Applied Boolean algebra*. 1969. (512.89 HOH Engin.) |
| Kiviat, Philip J. | *The Simscript II programming language*. 1968. (651.8 KIV Engin.) |
| Dakin, Christopher John. | *Circuits for digital equipment*. 1967. (621.381958 DAK Elect.) |
| *Computer Science* | 1969. (651.8 COM Engin.) |
| Pollack, Seymour V. | *A guide to PL/I*. 1969. (651.8 POL Engin.) |

## BULLETIN BAFFLER

Here is the solution to the final Baffler:

A simple exchange procedure such as

```
procedure exch (a,b)  integer a,b
begin integer temp;  temp: =a;  a:=b;  L1:b:= temp end
```

fails when faced with exch (i,A[i]) since i is given a new value before the address of A[i] is calculated for the instruction L1.

A general exchange procedure must calculate the addresses of both workspaces before altering either of them.

This may be accomplished by nesting an integer procedure within the main exchange procedure.

```
procedure exch (a,b);  integer a,b;
     integer procedure ex (a,b) integer a,b;
     begin integer temp;  temp:=a;  a:=b;  ex:=temp end;
L2:b:= ex (a,b);
end;
```

In this case the address of b in instruction L2 is calculated before integer procedure 'ex' is entered.

The following solution to Baffler No. 4 has been run successfully on the CDC 6600 at the Sydney Data Centre, and is offered to you as a point of interest.

```
THIS IS THE SOLUTION TO BAFFLER NUMBER 4
BEGIN INTEGER ARRAY A[1:4];
INTEGER I;
PROCEDURE EXCH (B,C);
INTEGER B,C;
          BEGIN
          INTEGER PROCEDURE ABC (B,C);
          INTEGER B,C;
                    BEGIN INTEGER Z;
                    Z:=B;
                    B:=C;
                    ABC:=Z;
                    END ABC;
          C:=ABC (B,C)
          END EXCH;
A[2]:=4;
A[4]:=3;
I:=2;
EXCH (I,A[I]);
OUTREAL (61,A[2]);
OUTREAL (61,A[4]);
OUTREAL (61,I);
A[2]:=4;
A[4]:=3;
I:=2;
EXCH (A[I],I);
OUTREAL (61,A[2]);
OUTREAL (61,A[4]);
OUTREAL (61,I);
END BAFFLER NO 4;
EOP
```

The results were:

```
+2.0000000000000≠+000    +3.0000000000000≠+000    +4.0000000000000≠+000
+2.0000000000000≠+000    +3.0000000000000≠+000    +4.0000000000000≠+000
```