UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

COMPUTER CENTRE BULLETIN

Vol. 4, No 8 & 9
6 September 1971

Editor:
Mrs Sarah Barry

JULY BULLETIN

The July Bulletin was incorrectly numbered as Vol. 4 No 6. It should have been numbered as No 7 and the Editor wishes to apologize to anybody who might have been confused by this. This edition combines the August and September issues into a single copy numbered 8 and 9.

PUNCHED CARD OUTPUT FROM PDP-10

The Computer Centre now provides a service to provide punched card output of PDP-10 ASCII files. The punched card output is, in fact, obtained by using the GE-225.

(a) Service Request

To obtain PDP-10 ASCII files in punched card form, the user must complete the form:

'Request for PDP-10 Card Output'

This is obtainable from the Centre and must be submitted to the Operations Supervisor when completed satisfactorily.

(b) Output Collection

The punched card output will be available for collection from the PDP-10 output shelves.

(c) Charges

Charges will be levied on the basis of size of the PDP-10 file to be punched, and will be entered as a miscellaneous charge against the user's PDP-10 project number.

Charge rates are:	University Departments	$0.60 per K words
	Government Departments	$1.20 per K words
	Non-Governmental Organizations	$1.50 per K words

NEW FORTRAN SYSTEM

A new FORTRAN system has been implemented on the PDP-10. This system comprises a new FORTRAN IV compiler, a new FORTRAN execution package, and a revised set of FORTRAN library routines.

The new FORTRAN has undergone extensive testing and checking in the Computer Centre over the past few months. Many errors have been found and corrected, and the Centre now considers the new system to be in a reasonably reliable state. The possibility of unknown errors always exists and so users are requested to check results carefully and report any errors via a programming consultation.

As detailed below, this FORTRAN system provides many new facilities to the user, including direct access processing of files and improved diagnostics, and corrects most of the errors reported in the present version of FORTRAN. However, because of the extended facilities available, the size of the FORTRAN execution package has increased with the result that some large programs (i.e. those presently requiring more than 22K of core) may exceed the available user core size of 24K when run under the new system. In order to minimize this increase in size, the NAMELIST facility has been removed from the new FORTRAN execution package. Users who have problems because of the size increase, or because of the removal of NAMELIST, are asked to contact the Computer Centre.

The new FORTRAN will be available to both remote terminal and batch users. The version numbers for the new system are as follows; the compiler will be V23-F3, and the execution package and library routines, LIB40, will be V30.

1. NEW FACILITIES

1.1 Direct Access

The sequential reading and writing of data in FORTRAN programs is familiar to most users. For direct access, i.e., in order to access a particular record, it was necessary to process each record in the file until the appropriate record was encountered. With the use of FORTRAN direct access statements, this is no longer necessary and a program is able to access the desired record directly.

Direct access programming allows a programmer to access any record within a file independent of the location of the previously accessed record within that file. Direct input/output is desirable when only a few records in a large file are to be accessed, or when a file is to read or written in a non-sequential manner, as in sorting.

Direct access applies only to data files with fixed-length records on the disk. Any fixed-length record file (whether formatted or unformatted) which has been written with FORTRAN may be read or updated non-sequentially.

To use a file in this manner, the file must first be defined by a DEFINE FILE statement and then records accessed by means of direct READ or WRITE statements.

(a) DEFINE FILE

The structure of the file must be specified by means of the DEFINE FILE statement. This statement must appear in the program prior to any READ or WRITE on the file. The format is as follows:

```
DEFINE FILE (u, s, v, name, pj)
```

where the parameters u, s, v, name and pj are as described below.

u	This is the logical unit number. It must refer to one of the Fortran logical unit numbers 10, 11, 12 or 13.
s	This is the size of the fixed length records within the file. For ASCII files (i.e. formatted files), the size is specified by the number of characters per record and can vary from 1 to a maximum of 132 characters. For binary files (i.e. unformatted files), the size is specified by the number of words per record and can vary from 1 to any number depending on the limitations of available space.

v	This is the associated integer variable. It always contains a value one greater than the number of the last record read or written.
name	This is the name of the file. If it is zero, then the standard default name of FORu (where u is the unit number) is assumed.
pj	This is the project number of the person whose disk area is to be accessed. It is not possible to create a file on another project area, but only to access an existing file for reading or updating, depending on the permission set by the owner. If the project number is zero, or omitted, the user's own project number is accessed.

IFILE and OFILE are not required to describe further a file defined in the above manner, DEFINE FILE is sufficient.

examples:

(i)

```
DEFINE FILE (1Ø, 15, IVAR, 'DATIN')
```

This statement defines a file assigned to logical unit 10. The name of the file is DATIN. If it were an ASCII file, then the records would be 15 characters long; if a binary file, then the records would be 15 words long. IVAR is the name of the associated integer variable.

(ii)

```
NAMFIL  =  'FILD'
DEFINE FILE (12, 2ØØ, INO, NAMFIL, 37)
```

Similarly, this defines a file called FILD assigned to logical unit 12. The file is a binary file (ASCII files cannot contain records exceeding 132 characters in length) with records of 200 words. It exists on the disk area belonging to project 37.

(b) Direct READ and WRITE Statements

Direct access READ and WRITE statements are differentiated from sequential I/O statements by the presence of the single quote (') following the logical unit number. Each must specify also the record number at which reading or writing is to start. The following formats are possible:

formatted I/O	unformatted (binary) I/O
READ(u'r,f) list	READ(u'r) list
WRITE(u'r,f) list	WRITE(u'r) list

where u is the logical unit number given in the DEFINE FILE statement;
r is the record number where I/O is to commence. This may be an integer constant, variable or expression;
f is the FORMAT statement number, and
list is the I/O list.

Notice that the logical unit number and the record number are separated by the quote sign and not by a comma.

The associated integer variable provides sequential access to records. To process a file sequentially, the program simply uses the value of the associated variable as the record number in the READ and WRITE statements.

```
        DEFINE FILE (14, 2Ø, INT, 'MSTER')
        .
        .
        .
        READ (14'INT)
        .
        .
        .
```

examples:

(i) To access the kth record of an ASCII file called DATER, coding might be

```
        DEFINE FILE (11, 25, IV, 'DATER')
        .
        .
        .
        READ (11'K, 5) A,B,C,I,J
    5   FORMAT (3A5, 2I5)
        .
        .
        .
```

Note that the size of the record in DEFINE FILE corresponds to the size of the FORMAT specifications.

(ii) Random WRITES are used to change every 7th record, beginning with record number 3 in the file named DATA on the user's disk area. The file is unformatted, contains 100 records, each 35 words long.

```
        DIMENSION LIST (35)
        DEFINE FILE (13,35,IVAR, 'DATA')
        .
        .
        .
        DO 2Ø K=3, 1ØØ,7
        WRITE (13'K) LIST
        .
        .
        .
    2Ø  CONTINUE
        .
        .
        .
```

If a direct READ or WRITE statement is followed by a sequential READ or WRITE statement on the same logical unit, then reading or writing begins with the next sequential record.

Note that it is inadvisable to create a file in the first instance or enlarge a file further ahead than the next sequential record using direct access writes. A file to be directly accessed should initially be created by sequentially writing the full number of blank records required.

1.2 END and ERROR

It is now possible to specify in a READ statement the desired transfer of control should an end-of-file or an error condition be encountered during a formatted, direct or sequential read operation. The errors trapped in this way do not include transmission or parity errors.

The format of the statement is

```
          READ (u,f, END=n, ERR=m) list
    or    READ (u'r,f, END=n, ERR=m) list
```

where u,f, and list are defined as usual (i.e., the unit number, the number of the FORMAT statement and the I/O list of variables), and n and m are the statement numbers to which program control is to be transferred should an end-of-file or error condition, respectively, occur.

example:

```
          READ (5,29,END=999, ERR=525) A,B,C
          .
          .
          .
     525  (control will transfer here should an error condition arise
           during the read operation)
          .
          .
          .
     999  (control will transfer here should the end-of-file be
           encountered by the read operation)
```

The arguments END=n and ERR=m are both optional. Both or either may be included and, if both are present, the order of precedence is unimportant.

If either an end-of-file or an error condition is encountered, then control will pass to the statement number declared by END=n or ERR=m. Should the appropriate parameter not be specified in the READ statement, then the execution of the user's program will be terminated and an appropriate error message will be printed.

1.3 ENCODE and DECODE

The ENCODE and DECODE statements transfer information from one data area to another, converting the data according to a given Format statement in the process. DECODE is used to change data in ASCII character format to data in some other form, and ENCODE changes data from the various internal representations into data in ASCII character form.

The format for these two statements is as follows:

```
ENCODE (c,f,r) list
DECODE (c,f,r,) list
```

where c is the number of ASCII characters in the character string,
f is the FORMAT statement number,
r is the starting address of the ASCII character string referenced, and
list is the I/O list of variables.

examples:

(i) Suppose A(1) contains the binary number 300.45, A(2) the binary number 3.0, J a binary integer 1, and B is a four word array. Then the statements:

```
      DO 3Ø J=1,2
      ENCODE (16,2Ø,B) J,A(J)
  2Ø  FORMAT (1X, 'A(',I1,')Δ=Δ', F8.2)
      .
      .
      .
  3Ø  CONTINUE
```

would cause the array B, after the first iteration of the DO loop, to contain the character string 'ΔA(1)Δ=ΔΔΔ3ØØ.45' That is, the contents of each element of B would be:

```
B(1)   ΔA(1)
B(2)   Δ=ΔΔΔ
B(3)   3ØØ.4
B(4)   5
```

After the second iteration of the loop, the array would contain:

```
B(1)   ΔA(2)
B(2)   Δ=ΔΔΔ
B(3)   ΔΔ3.Ø
B(4)
```

(ii) Suppose also that C contained the ASCII string 35279, then the following statements:

```
      DECODE (4,15,C) B
  15  FORMAT (2F1.Ø, 1X, 2F1.Ø)
```

would cause the first two characters of C (3 and 5) to be converted to floating point binary values and stored in B(1) and B(2); the next value of C to be skipped; and the last two values of C (7 and 9) to be converted and stored in B(3) and B(4).

The following program demonstrates some uses for ENCODE and DECODE.

```
      DIMENSION UNPK(5),FMT(6)
C     SOME EXAMPLES OF ENCODE AND DECODE
C
C     *****TO UNPACK ASCII CHARACTERS*****
      PACKED='ABCDE'
      DECODE (5,2Ø,PACKED) UNPK
2Ø    FORMAT(5A1)
      PRINT 3Ø,PACKED,UNPK
3Ø    FORMAT('PACKED = ',A5,'UNPACKED TO ',5A2)
C
C     *****TO SELECT A CHARACTER FROM A WORD*****
      FLAGS='MBCRF'
      DECODE(5,1Ø,FLAGS) BUSY
1Ø    FORMAT(1X,A1,3X)
      PRINT 4Ø,BUSY
4Ø    FORMAT(/' THE BUSY FLAG IS ',A1//)
C
C     *****TO CHANGE THE SECOND CHARACTER OF "FLAGS" TO BLANK*****
      OFF=' '
      ENCODE(5,1Ø,FLAGS)OFF
      DECODE(5,1Ø,FLAGS)BUSY
      PRINT 4Ø,BUSY
C
C     *****TO SET UP A RUN TIME FORMAT ARRAY*****
      NUMBER=9
      ENCODE(27,5Ø,FMT) NUMBER
5Ø    FORMAT('(''MOVING ARROW '',',I3,'X''↑'')')
      PRINT 6Ø,FMT
6Ø    FORMAT(' THE FORMAT IS ',6A5)
      PRINT FMT
      END
```

During execution, the program produces the following results.

```
PACKED = ABCDE UNPACKED TO A B C D E

THE BUSY FLAG IS B


THE BUSY FLAG IS

THE FORMAT IS (' MOVING ARROW ', 9X,'↑')
MOVING ARROW          ↑
```

1.4 Multiple Returns from Subroutines

In both Function and Subroutine subprograms, it is possible for the subprogram to return to the main program at an address other than that immediately following the call to the subroutine.

This can be done in the following way:

Statement labels can be specified as arguments to a subroutine by preceding them in the argument list by an asterisk (*) or a dollar sign ($). The corresponding dummy argument in the subroutine statement must be either a $ or a * sign.

Within the called subprogram, the return to the main program is effected by a new form of the RETURN statement.

```
             RETURN i
```

where i is an integer constant or variable. The value of i must be positive, and specifies that the return is to the ith argument of the argument list of the subprogram (where the ith argument is a statement number preceded by a dollar or asterisk sign). If i=0, the return made is the same as with the normal RETURN statement.

examples:

```
(i)          CALL TYPE (A, $1Ø, B, $2Ø)
             .
             .
             .
        1Ø   - - -
             .
             .
             .
        2Ø   - - -
             .
             .
             .
             END
             SUBROUTINE TYPE (V1, $, V2, S)
             .
             .
             .
             RETURN NUM            ;  If NUM = 2, return is to statement
             .                          number 10 in the main program,
             .                        If NUM = 4, return is to statement
             .                          number 20 in the main program
             RETURN                ;  This is the normal return and will
             END                        return to the statement following
                                        the subprogram call
```

```
(ii)          .
              .
              .
              K = LIST (I, $93, J)
              .
              .
              .
              END
              FUNCTION LIST (N, $, M)
              .
              .
              .
              RETURN 2          ;  for an error condition, say, returns to
              .                      statement 93 in the calling program
              .
              .
              RETURN            ;  for normal completion
              END
```

When a RETURN i is used (where i is not equal to zero) in a Function subprogram, the value returned in the name of the function is lost.

The use of a dollar sign is preferred since expressions involving the multiplication sign (*) can be used as arguments in subprogram calls.

The modification to the PLOTI subroutine (section 5.3) provides an illustration for multiple returns.

1.5 Output Field Exceeds Format

With I, F, O and D type formats, the operating system will print all asterisks in the field when the number to be output exceeds the size of the field defined in the FORMAT specification.

example:

```
              I = 34
              J = 9376
              A = 126.527
              B = 52.35
              .
              .
              .
              WRITE (6,1Ø) I,A
          1Ø  FORMAT ('ΔΔINTEGERΔISΔ',I3,',ΔREALΔISΔ',F5.2)
              .
              .
              .
              WRITE (6,1Ø) J,B
              .
              .
              .
```

would produce the following results:

```
        ΔINTEGERΔISΔΔ34,ΔREALΔISΔ*****
        ΔINTEGERΔISΔ***,ΔREALΔISΔ52.35
```

Note that G type formats should be used if there is any uncertainty about maximum field width required.

The execution summary will include a count of output field width overflows if any occur.

1.6 JOBBAL Function

JOBBAL is a FORTRAN IV function that has been added to the library. It returns to the user program the remaining balance of the job cost limit.

At present, with the use of the JOB and/or the LIMIT commands, a user imposes a cost limit on a program. When the limit is exceeded, the program execution is terminated. With the use of the JOBBAL function, a program can control itself by examining the balance left and terminating itself cleanly should there be insufficient funds available.

The JOBBAL function returns to the calling program an integer number of units. The value of a unit is 1 cent for university users, 2 cents for government departments and 2.5 cents for other users. This means that a given number of units represents a constant amount of computing for each class of user.

example:

```
        .
        .
        IBAL = JOBBAL (Ø)
        IBAL = IBAL * 2
        IF (IBAL.LE.2Ø) GO TO 999
        .
        .
        .
```

1.7 New Type Declaration Statement

A new type declaration statement, SUBSCRIPT INTEGER, is now available. This allows for the declaration of fixed point variables that fall in the range -2^{27} to 2^{27}.

1.8 Dollar Sign in Format

A dollar sign ($) as a format field specification code suppresses the carriage return at the end of the Teletype or line printer line.

1.9 ERRSET Function

ERRSET allows the user to control the printout of execution-time arithmetic error messages (see section 2.1). ERRSET is called with one argument in integer mode.

```
        CALL ERRSET (N)
```

Printout of each type of error message is suppressed after N occurrences of that error message. If ERRSET is not called, the default value of N is 2.

2. EXECUTION DIAGNOSTICS AND SUMMARY

2.1 Execution Diagnostics

These error messages are diagnostics produced by the FORTRAN operating system during execution of a program.

(a) These messages are all followed by a second message 'LAST FORTRAN I/O AT USER LOC adr'.

DEVICE dev: NOT AVAILABLE

> The operating system tried to initialize a device which either does not or has been assigned to another job.

DEVICE NUMBER n IS ILLEGAL

> A non-existent device number was selected.

END OF FILE ON dev:

> A premature end-of-file has occurred on an input device.

FILE NAME filename NOT ON DEVICE dev:

> The file cannot be found in the directory of the specified device.

ILLEGAL CHARACTER, x,IN FORMAT

> The illegal character x is not valid for a FORMAT statement.

INPUT DEVICE ERROR ON dev:

> A data transmission error has been detected in the input from a device.

ILLEGAL CHARACTER, x,IN INPUT STRING

> The illegal character x is not valid for this type of input.

NO ROOM FOR FILE filename ON DEVICE dev:

> There is no room for the file in the directory of the named device or no room on the device.

program name NOT LOADED

> A dummy routine was loaded instead of the real one. Generally, this error occurs when a loaded program is patched to include a call to a library program which was not called by the original program at load time.

OUTPUT DEVICE ERROR ON dev:

> A data transmission error has been detected during output to a device.

PARITY ERROR ON dev:

A parity error has been detected.

REREAD EXECUTED BEFORE FIRST READ

A reread was attempted before initializing the first input device.

dev: WRITE PROTECTED

The device is WRITE locked.

(b) These messages are all followed by a second message 'LOADING OVERLAY name FROM LOCATION adr'.

OVERLAY NUMBER INCORRECT

A call to overlay with a number 0 or greater than 20.

OVERLAY NOT IN TABLE

The name in the overlay call does not exist.

ERROR READING OVERLAY FILE

OVERLAY WILL OVERWRITE CALLER

FILE NOT FOUND

(c) These messages are not followed by another message.

ACOS OF ARG > 1.Ø IN MAGNITUDE

ASIN OF ARG > 1.Ø IN MAGNITUDE

ATTEMPT TO TAKE SQRT OF NEGATIVE ARG

CLOSE FAILURE FOR PLOTTER FILE

* FLOATING DIVIDE CHECK PC=nnnnnn

* FLOATING OVERFLOW PC=nnnnnn

* FLOATING UNDERFLOW PC=nnnnnn

* INTEGER DIVIDE CHECK PC=nnnnnn

* INTEGER OVERFLOW PC=nnnnnn

OPEN FAILURE FOR PLOTTER FILE

X COORDINATE OUT OF BOUNDS
This is a plotting error. The y coordinate may also be out of bounds.

Y COORDINATE OUT OF BOUNDS
The x coordinate will have been tested first, and is therefore within bounds.

* These error messages are typed for each occurrence of the appropriate error for a maximum number of times. This maximum number is set by default to 2, but can be changed by means of the ERRSET function (see section 1.8).

2.2 Execution Summary

At the end of execution of a program, a summary will be printed that lists the actual number of times each error message occurred. The execution time and total elapsed time for the run are also given.

The possible errors accounted for in the summary are:

```
ACOS OF ARG > 1.Ø IN MAGNITUDE
ASIN OF ARG > 1.Ø IN MAGNITUDE
ATTEMPT TO TAKE SQRT OF NEGATIVE ARG
FATAL I/O ERROR
FLOATING DIVIDE CHECK
FLOATING OVERFLOW
FLOATING UNDERFLOW
INTEGER DIVIDE CHECK
INTEGER OVERFLOW
OUTPUT FIELD WIDTH OVERFLOW
OVERLAY ERROR
PLOTTER ERROR
```

examples:

(i)

```
EXECUTION TIME:                     Ø.16 SEC.
TOTAL ELAPSED TIME:                17.8Ø SEC.
NO EXECUTION ERRORS DETECTED.
```

(ii)

```
EXECUTION TIME:                     Ø.24 SEC.
TOTAL ELAPSED TIME:                 3 MIN. 26.64 SEC.

NO. OF ERRORS                       ERROR TYPE

     1                           INTEGER OVERFLOW
     4                           OUTPUT FIELD WIDTH OVERFLOW
```

3. REPORTED ERRORS CORRECTED IN THE NEW VERSION OF FORTRAN

(a) Expressions involving a mixture of variable types are better handled by the compiler.

In the error reported in the Bulletin Vol. 3, p. 49, i.e.,

```
A = X**(I1-I2+I3)
```

the sub-expression is now evaluated as an integer and the real-integer exponentiation routine used.

In the Bulletin Vol. 4, p. 94, the expression

```
D = S*D/(2*I-1)
```

is reported to be translated incorrectly. The correct code is now produced.

(b) Implicit conversion from double precision to real when the number is almost a power of 2 is now accurate (see Bulletin Vol. 2, p. 106).

(c) Octal constants greater than 2^{35} may be defined in assignment statements,

```
B = "777777ØØØØØØ
```

(see Bulletin Vol. 2, p. 128).

(d) Correct code is produced when a function is used in the index expression for double precision or complex arrays (see Bulletin Vol. 2, p. 128).

(e) The complex expression

```
Z = Z1/2
```

is also handled correctly (see Bulletin Vol. 3, p. 38).

(f) The use of a variable name as both a simple integer and as a function name now produces a compilation error message.

```
IJK = 92
L = J + IJK(2,3)
```

(See Bulletin Vol. 3, p. 94.)

(g) A logical IF may not compare a complex and a real variable. For example,

```
IF (C(J).GT.1.5) GO TO 1ØØ
```

will now produce an error message. (See Bulletin Vol. 3, p. 93.)

(h) A literal constant may not consist solely of two adjacent single quotes ('), for example, B = ''

This will produce a compilation error (see Bulletin Vol. 2, p. 129).

(i) The differences between the truncation of negative real values in PDP-10 FORTRAN and GE-225 FORTRAN, reported in Vol. 3, pp. 49-50 of the Bulletin, no longer apply. The routine IFIX, and all implicit fixing of real variables now use the INT method of truncation towards zero instead of towards minus infinity. See section 5.1

(j) The compiler generates code to restore the DO loop index when statements could extend the range of the DO loop. This corrects errors reported in the Bulletin Vol. 3, p. 73 and Vol. 4, p. 93.

(k) Double precision output has been corrected, and the comment on values outside the range $0.1*10^{-16}$ to $0.1*10^{8}$ given in the Bulletin Vol. 3, p. 54 no longer applies.

(l) A mixture of H type and single quote (') type Hollerith strings in FORMAT statements is now allowable (see Vol. 4, pp. 38-40).

(m) The 026 character ')' which was not accepted is now converted to the 029 ')' on input, as are the other 026 characters correspondingly converted (see Bulletin Vol. 4, p. 63).

(n) Further efforts have been made to solve the problems caused by Batch suppressing trailing blanks. Some improvement has been made, but A type format still appears to have problems (Vol. 3, pp. 39-40).

(o) Some additional errors corrected are:

   (i) RELEASE now clears all the flags it should.

   (ii) Backspacing records in binary and ASCII disk files now works properly.

   (iii) Tabs in format statements are treated as spaces. Previously, tabs were illegal unless they were within a Hollerith string.

   (iv) Backpointing of T format type on input is now correctly handled.

(p) Other changes in the FORTRAN operating system are:

   (i) A negative argument to SQRT now returns the square root of the absolute value instead of zero as well as giving the error message.

   (ii) Floating point underflow and overflow, integer underflow, overflow and dividing by zero produce error messages. For floating point operations, the result produced is zero for underflow and $.17 \times 10^{39}$ for overflow and divide checks.

   (iii) The FORTRAN operating system now uses FRECHN UUO to allocate channels. Any user-written MACRO I/O routines should also use this UUO to avoid clashes in channel allocations.

Users are reminded that a list of all current errors is kept in one of the blue binders in the Clients' Room. This list is kept up to date and is intended to provide quicker reference on errors than searching through all the latest Bulletins and Newsletters.

4. KNOWN ERRORS IN THE NEW FORTRAN

(a) Double precision primary to integer primary conversion may cause problems. Precision of the result is limited to 8 decimal digits.

(b) IFIX may cause truncation errors for very large numbers.

(c) Oversize formats containing slash, and all oversized integer formats, cause records to be skipped.

(d) Oversize format for ENCODE statement causes '?ILL MEM REF' error message.

(e) Specifying a character count too long or too short in ENCODE or DECODE is not diagnosed as an error.

(f) The last digit of E and F-type output with a negative scaling factor is often incorrect.

5. FURTHER ITEMS OF INTEREST

5.1 IFIX, INTIER and ENTIER Functions

The result of an IFIX function is now the same as INT. That is, it converts a real number to an integer and the result given is

sign of arg * largest integer $\leqslant |\text{arg}|$

Users who might still wish to use IFIX as it was previously defined, can use INTIER instead. This converts a real number to an integer and the result given is

largest integer $\leqslant$ arg

There is a new function ENTIER which performs in much the same way as INTIER, except that the result is real and not integer. That is, it converts a real number to real and the result given is

largest real $\leqslant$ arg

example:

The results of IFIX, INTIER, ENTIER for a group of arguments are as follows:

argument	IFIX	INTIER	ENTIER
2.Ø	2	2	2.Ø
1.5	1	1	1.Ø
1.Ø	1	1	1.Ø
Ø.5	Ø	Ø	Ø.Ø
Ø.Ø	Ø	Ø	Ø.Ø
-Ø.5	Ø	-1	-1.Ø
-1.Ø	-1	-1	-1.Ø
-1.5	-1	-2	-2.Ø
-2.Ø	-2	-2	-2.Ø

5.2 REWIND Statement

If a REWIND is used on a disk file, then any prior assignment of a named disk file to a logical unit number will be broken. If the file is a scratch file, an automatic reassignment will be made to the file by using the same unit number. If a named data file that has been assigned with IFILE or OFILE is rewound, then the file must be assigned by a further call to IFILE or OFILE after the REWIND.

Users are recommended to use ENDFILE rather than REWIND.

5.3 Modification to PLOTI Subroutine

An additional optional argument has been added to the calling sequence of PLOTI. This argument, if present, is in the form of $n where n is a statement number to which control will pass if any of the subsequent plotting routines fail. Thus, 'X COORDINATE OUT OF BOUNDS' need not be a fatal error for the program, although that plot file is closed and a new call to PLOTI should be the next plotting operation. (For this purpose, a call to WHERE is not regarded as a plotting operation).

example:

```
              CALL PLOTI ('GRAPH', 1Ø.Ø, $2ØØ)
              .
              .
              .
         2ØØ - - -                 , returns here on any plotting error
```

NEW BATCH SYSTEM

A new batch processing system will be implemented shortly on the PDP-10. The new batch provides users with a wider range of facilities, but does introduce some differences to the earlier batch operation.

1. NEW FACILITIES

With a few exceptions, all the facilities available to remote terminal users are now available to batch users. For full details of available facilities and services, users are referred to the System User's Guide and Computer Centre Bulletins Vol. 4 Nos 1-7.

The exceptions (detailed below) arise because of the non-interactive nature of batch processing.

1.1 Commands

All commands detailed in Chapter 6 of the System User's Guide are now available via Batch, and operate as described.

In respect of commands the following points should be noted.

(a) The default job input and job output devices are the card reader and line printer.

(b) LOGIN and FINISH do not operate via batch. Their equivalents are JOB and EOJ.

(c) EOJ automatically deletes any files from the user's area that have not been specifically KEPT. If there are too many KEPT files, EOJ deletes enough files to get below the LIMIT starting with the youngest files first.

(d) The TYPE command is not available via Batch. The equivalent function can be obtained by the COPY command using the default job output device.

example:

```
            .COPY FROM=RANFIL        ; TO the line printer
                                       is assumed.
```

(e) Because of its detailed interactive nature, DDT is not suitable for Batch. It is recommended that users exercise extreme caution in any attempt to use DDT via Batch.

(f) The LIMIT command can only be used to alter the cost limit for a task, phase of tasks or a job while the batch job is running. It cannot be used to reset the job cost limit when that limit is exceeded. In Batch, exceeding the job cost limit will result in automatic termination of the job.

1.2 Differences to Previous Batch

There are three important command changes which the new batch system will introduce.

(a) The default option in a FORTRAN command is now NOLIST (see System User's Guide section 6.4.12). Thus .FORTRAN will not produce a listing. If an output listing is required it must be specifically requested with the LIST option

i.e. .FORTRAN(LIST)

(b) The default option in a RUN command is now NOMAP. If either a MAP or SYMBOL map is required they must be specifically requested (see System User's Guide section 6.4.21).

(c) The OVERLAY command now uses AREA and NAME as options. Hence these must be enclosed in round brackets.

```
.OVERLAY AREA = 1, NAME = FRST
```

under the earlier batch, now becomes

```
.OVERLAY(AREA=1, NAME=FRST)
```

(see System User's Guide section 6.4.17).

This allows specification of Files to be included on the OVERLAY command.

1.3 File Storage

The permanent file storage allocated to each batch project is 12.5 Kwords. This will enable the batch user to keep relocatable binary program files in the system. Hence programs need not be recompiled on each run.

In addition, a job, while running, will have available 128 Kwords of scratch file space.

2. DECK SETUP

The deck setup required for a batch job is the same as at present, and is documented in Chapter 7 of the System User's Guide.

3. CHARGES

The charges for batch processing are now levied on the same basis as for remote terminal processing. Thus batch work is charged in accordance with the schedule of charges given in Appendix B.2 of the System User's Guide.

While this increases the number of categories for which charges are made, users will find that because of the new facilities available (e.g. keeping binary program files on disk) overall computing costs should be substantially reduced.

4. NEW BATCH AND REMOTE TERMINAL USERS

With the implementation of the new Batch, remote terminal users will be able to access their files from both their terminal or through Batch. Program files can be created via Batch, debugged via the terminal and then run from Batch. The permanent file storage space available to terminal and batch projects is 37.5 Kwords.

As all line printer output (from both Batch and remote terminals) is now controlled by the line printer symbiont, remote terminal users should experience improved availability of their printer listings.

Although this new version of Batch has been extensively tested, there is always the possibility that undetected errors still exist in the system. Users are requested to check their results carefully and report any significant discrepancies to the Centre.

NEW COBOL OPTIONS

Two new options are now available for COBOL and they work through terminals and the new Batch. These options are STD and NONSTD and they refer to the presence or absence of sequence numbers in a source program.

STD is the COBOL standard, i.e. source programs are assumed to have sequence numbers. This is a default option.

NONSTD implies that there are no sequence numbers in the source program. This is generally the case with programs prepared via Teletypes. NONSTD can be abbreviated to NS.

The command format for COBOL is as follows:

```
           BIN    LIST              STD
     COBOL(NOBIN,NOLIST,MACRO,MAP,NONSTD)

     {IN=}filename-1,{BIN=}filename-2,{LST=}filename-3

filename-1 is the name of the source file
filename-2 is the name of the resulting relocatable file
filename-3 is the name of the list file
```

TERMINATION OF FILE CREATION SERVICE

Since the release of remote terminals in January this year, the Centre has provided a service to create disk files from terminal users' card decks. As the run can now create files via the new batch system, the File Creation Service has been discontinued.

LIBRARY ACCESSIONS

NATIONAL MEDICAL AUDIOVISUAL CENTER	*Computer printout; selected list of audio-visuals* 1970 (Qto 016.6138 NAT Cent Med.)
JAHODA, Gerald	*Information storage and retrieval systems for individual researchers* 1970 (Z695.9.J35 Main)
JOHNSON, Albert Frederick	*A programmed course in cataloguing and classification* 1968 (Z695.J673 Main)
LANDAU, H.B.	*Research study into the effective utilization of machine-readable bibliographic data bases* 1969 (Qto Z699.L37 Main)
DORFMAN, Robert	*Linear programming and economic analysis* 1958 (330.182 DOR Arch.)
INTERNATIONAL SYMPOSIUM ON OPTIMIZING AND ADAPTIVE CONTROL. 1st Rome, 1962	*Proceedings* 1962 (Qto 519.92 INT Engin.)
U.S. National Bureau of Standards. Computation Laboratory.	*Tablitsy veroiatnostnykh funktsii* 1970 (Qto 519.0835 UNI Maths.)
THE COMPUTER SOCIETY OF CANADA. National conference	*Proceedings.* 5th 1966 and onwards (QA76.5.C613 Engin.)
APTER, Michael J.	*The computer simulation of behaviour* 1970 (BF39.5.A65 Main)
NATIONAL COMPUTING CENTRE	*Computer application packages in local government* 1969 (621.38195 NAT Engin.)
	Computer design. v.8 1969 and onwards (TK7888.3.C65 Engin.)

EVANS, A.J.	*Periodicals data automation project* 1969 (Qto Z695.7.E85 Main)
PARSLOW, R.D. ed.	*Advanced computer graphics* 1971 (Qto 001.53 PAR Engin.)

ACM SYMPOSIUM ON PROBLEMS IN THE OPTIMIZATION OF DATA PROCESSING SYSTEMS Pine Mountain, Ga. 1969
Proceedings 1969 (Qto 651.8 ACM Engin.)

ACM SYMPOSIUM ON THEORY OF COMPUTING, Marina del Ray 1969
Conference record 1969 (Qto 510.7834 ACM Engin.)

COCKE, John	*Programming languages and their compilers* 1970 (Qto 651.8 COC Maths.)
MINORSKY, Nicholas	*Theory of nonlinear control systems* 1969 (629.836 MIN Elect.)

PRINCETON SYMPOSIUM ON MATHEMATICAL PROGRAMMING Princeton University 1967
Proceedings 1970 (519.92 PRI Biol.)

Computers in construction communications 1970 (Qto 690 COM Arch.)

BATSTONE, Druce Barry	*Solution to recycle problems in computer-aided design* 1970 (THE 4208 Main)

COMMAND CARDS

Users are reminded that the PDP-10 command cards for Batch must be punched on the 029 card punches and not the 026 card punches.

The FORTRAN compiler and operating system have been patched to accept special characters in either code. However, command cards contain commands issued to the PDP-10 monitor which is not a part of the FORTRAN system and thus will not accept 026 code.

MANUALS

A number of users appear to be unaware of the manuals that the Centre has produced, or where they can be obtained.

At present, all manuals can be bought from the University Bookshop and the manuals that they should have in stock are as follows:

FORTRAN (1st Revision is incorporated in the manual; 2nd Revision is obtainable separately at no cost)	MNT-5	$1.50
EDITOR	MNT-6	$1.00
SYSTEM USER'S GUIDE	MNT-8	$3.90
BASIC	MNT-9	$2.70
AID	MNT-10	$4.20
UTILITY PROGRAMS (Absolute Overlays) (Digital Plotter)	MNT-12	$1.20

RANDOM NUMBER GENERATING SUBPROGRAMS

The details of RAN, SAVRAN and SETRAN are described incorrectly in the PDP-10 FORTRAN manual MNT-5. Their correct description is as follows.

RAN is a function subprogram which generates single precision random numbers in the range $0 < x < 1.0$

```
        e.g.  VAR  =  RAN(Ø)
```

Note that the value of the argument is ignored.

SAVRAN and SETRAN are subroutine subprograms required to service RAN.

SETRAN is used to provide a non-standard starting point for RAN.

```
        e.g.  CALL SETRAN (K)
```

where K has a value in the range $0 < K < 2^{31}-1$. The standard starting point is 524287. Note that if SETRAN is not used RAN will return the same set of 'random' numbers each time the program is run.

SAVRAN is used to save the integer which would be used by the next call to RAN. Thus a sequence of 'random' numbers produced by RAN can be regenerated if the starting point has been saved.

example:

```
          CALL  SETRAN(K)              ; sets a non-standard start for RAN
          DO I∅   I  = 1,N
          X  =  RAN(∅)                 ; generates some random numbers
          . . .
     1∅   CONTINUE
          CALL SAVRAN (NUMBER)         ; save the next starting value
          DO 2∅   I  =   1,N
          Y  =  RAN(∅)
          . . .
     2∅   CONTINUE
          CALL SETRAN (NUMBER)         ; insert the start to reproduce the
                                         previous set of 'random' numbers.
          DO  3∅   I   =   1,N
          Y  =  RAN(∅)                 ; gives the same set as previously.
          . . .
     3∅   CONTINUE
```

To produce a random integer in a given range the following could be used

```
          K   =   1∅∅∅*RAN(∅)
```

The following sample program uses SETRAN with a non-reproducible argument. RAN may then be called to produce a series of random numbers.

```
          INTEGER HRS
          CALL TIME (NOW)
          DECODE (5,1∅,NOW) HRS, MINS
     1∅   FORMAT (I2,1X,I2)
          K = MINS*1∅∅ + HRS
  C       THIS SCRAMBLES THE TIME
          CALL SETRAN (K)
  C       THE STARTING VALUE DEPENDS ON THE
  C       TIME OF DAY AND IS NOT PREDICTABLE
          . . .
          X = RAN(∅)
          . . .
```

CHANGES TO LOGOUT

There have been some improvements made to the logging out procedure.

(a) Error Messages

If logout encounters problems with a user's files an error message is given to the user and logout continues, instead of giving a fatal stop as at present.

The error messages could be any of the following:

(i) FILE DIRECTORY NOT FOUND

Logout has not been able to find the user's file directory, and cannot reference any of the files. Therefore, no file processing is done.

(ii) ERROR READING FILE DIRECTORY

Logout has encountered an error trying to read the user's file directory. No further file processing is done.

(iii)
```
                    FILE NOT FOUND
OPEN                DIRECTORY NOT FOUND
KEEP    FAILURE (   PROTECTION FAILURE )    FOR FILE filename
DELETE              FILE BEING MODIFIED
                    FILE NOT OPEN FOR RENAME
```

These are a variety of messages which can appear for individual files within the directory. This particular file will not be processed but logout will continue processing any further files on the directory.

(b) New User Question

If many users are logging in or logging out at the same time, the accounting files can become busy. In this case, the user receives the message

```
THE ACCOUNTING FILES ARE BUSY

DO YOU WISH TO CONTINUE WITH LOGOUT?    (ANSWER Y OR N):
```

Answer Y <cr> if you wish to continue with logout

N <cr> or ↑C if you wish to abort logout

PLOTTER OUT OF BOUNDS

It has been found that points apparently within the valid area for plotting sometimes give the 'Plotter Coordinates out of bounds' message. This occurs only in the immediate vicinity of the boundaries and appears to be caused by rounding errors in the calculation of the pen position. The Centre is currently working on this problem.

In the meantime it is suggested that users limit their plotting so that they do not attempt to use the area within 1/10" of any of the boundaries.

A simple way of doing this would be to move the origin immediately after opening the plot file using the plot command

```
CALL PLOT (Ø.1, Ø.1, -3)
```

carefully controlling the pen movement at the boundaries in the positive X and Y directions.

MATINV ERROR

There is an error in the MATINV subroutine (classification number D4.205 for the GE-225 and D4.505 for the PDP-10).

The tolerance test in MATINV attempts to divide by zero in some cases, particularly with sparse matrices. The error occurs in the second statement before label 140 which currently reads

```
IF(ABS(T3)/(ABS(T1)+ABS(T2)).LT.TOL)T3=Ø.Ø
```

This statement should be replaced by

```
IF(ABS(T3).LT.(ABS(T1)+ABS(T2))*TOL)T3=Ø.Ø
```

The version obtainable from the Computer Centre has now been updated to correct this error.

EIGEN AND ZEIGEN SUBROUTINES

Ian Oliver

The subroutine EIGEN and ZEIGEN for finding the eigenvalues and eigenvectors of arbitrary real matrices are now available for the PDP-10.

The calling sequences have been changed from the GE-225 version to eliminate problems caused by a compiler bug (soon to be fixed). In addition a tolerance value has been changed to reflect the PDP-10 word size.

New writeups and decks are available from the Computer Centre for EIGEN (D4.569), ZEIGEN (D4.570) and a test input/output program TEST (D4.570).

The first card of ZEIGEN has been changed to read:

```
SUBROUTINE ZEIGEN (EIGVAL, EIGVEC, A, K, L, T, NVECT, ITER)
```

Also, the statement declaring EIGVAL and EIGVEC to be complex has been placed in front of the statement dimensioning these arrays.

The following changes have also been made to the TEST program:

(i) Calling sequence changed as described above

(ii) The statement

```
        11Ø  FORMAT (2A3, 11A6)
```

has been replaced by

```
        11Ø  FORMAT (2A3, 11A5)
```

Note that the new EIGEN deck incorporates the tolerance value

EP = 7.451E-9

S M I S

R.D. Nilsson

The Department of Civil Engineering's Symbolic Matrix Interpretative Scheme (SMIS) which is available to PDP-10 users provides the equivalent of a desk calculator for matrix operations.

The program was originally obtained for the University of California (Berkeley) and has been extensively modified to provide free field input and interactive action. It is written in FORTRAN IV with some MACRO subroutines and can be operated via Batch or a remote terminal. A full write-up of the program is available from the Civil Engineering Department.

CIVIL DEPARTMENT USAGE

SMIS is mainly used in the department for the teaching of matrix structural analysis. Here we are interested in the formulation of the problem, but the arithmetical operations are too formidable unless the problems are trivial. This program allows, and in fact requires, the student to formulate the problem completely, but removes the arithmetical labour.

GENERAL FEATURES

SMIS is an example of a problem oriented language where each input line of data specifies an operation to be carried out and the data to be used in the operation. This data includes symbolic names of matrices previously stored.

For example, the matrix equation AX = B is to be solved where

$$A = \begin{bmatrix} 17.28 & 3. & -1. \\ 7.16 & 21.4 & 2. \\ 2. & 1. & 7.8 \end{bmatrix} \quad X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \quad B = \begin{bmatrix} -1.6 \\ \phi \\ 12.4 \end{bmatrix}$$

The sequence of SMIS commands could be:

```
START     *   SOLVE  AX = B
LOAD  A  3  3
17.28  3.  -1.    7.16  21.4  2
2      1   7.8
PRINT A  *  FOR CHECKING PURPOSES
DUPL  A  SA  *  SAVE  A  IN  SA  FOR  LATER
INVERT  A
PRINT  A     *  TO  LOOK  AT  INVERSE
LOAD   B   3   1
-1.6   Ø   12.4
PRINT  B
MULT  A  B  X    *  NOTE  A  IS  NOW  INVERSE  A
PRINT  X  *  THE SOLUTION
MULT  A  SA  ONES  *  CHECK HOW GOOD INVERSE WAS
PRINT ONES  *  SHOULD BE A UNIT MATRIX
```

The basic matrix operations of addition, subtraction, multiplication, scalar multiplication, inversion and transposition are available and eigenvalues and eigenvectors can be obtained for real symmetric matrices. A number of submatrix operations are available plus a number of service operations such as generation of zero and unit matrices.

MATRICES

Storage in the PDP-10 allows approximately 20000 matrix elements distributed amongst up to 40 different matrices. Matrices are given names containing up to five characters. Initially only the minimum core space is allocated and this core space is expanded as space is required for extra matrices. Thus users with small problems do not have to pay for any more core space than is required to accommodate their problem.

INPUT

All input is free field with one command per line except that matrix elements can be input using as many lines as required, although it is convenient to input one row per line. The free field input allows for mixed alphanumeric and numeric fields and extensive error checking and reporting is carried out so that the program is not stopped by a data error.

The program can continue on with succeeding problems when an error is detected making it extremely suitable for stacking student programs in batch.

If desired, input can be obtained from one or more files and transfers can be made back and forth between commands on a file and on a Teletype or in a card deck. This allows the equivalent of subroutines to be stored in a file and operated on with different sets of data.

OUTPUT

Output of results is obtained with the PRINT command which determines an output format such that the maximum element in a matrix is output with seven significant figures. If required, an E type output can be obtained to give details of the smaller elements in the matrix.

All commands are normally echoed if run from Batch, but this feature can be turned on and off with an ECHO command.

If desired some or all output can be directed to a file.

SAVE AND RESTORE

Any or all matrices may be saved on a file and selective restoration made within the current program run, or at a later date if the file is kept. This is useful with large matrices that are to be reused and since individual elements may be altered, errors can be easily corrected and the problem rerun.

DYNAMIC DATA STORAGE

One data pool is managed dynamically for all the matrices and any extra arrays used in such routines as the invert and eigen routines. Matrix names, sizes and positions in the data pool are stored in a table which is searched by the command decoder to find the matrices mentioned in a command.

When a matrix is deleted (or superseded by a new matrix with the same name) it is squeezed out of the data pool and the table adjusted accordingly. New matrices are always added at the end of the pool.

Only one location (A(1)) is allowed for the data pool in the program. When space is required to expand the data pool, a MACRO subroutine is called to obtain it. This space is allocated past the normal end of the program (JOBFF), leaving enough space for FORTRAN to obtain its I/O buffers and if necessary the current core allocation for the program is expanded. The MACRO subroutine returns an index I such that A(I) is the location immediately before the start of the allocated data pool. Thus the Jth location of the data pool is A(I+J).

FREE FIELD ROUTINE

This routine is written in FORTRAN and reads the input lines in 16A5 format. A MACRO subroutine is used to extract characters from the input array and a table of words built up by scanning the array.

These may be alphanumeric, integer or real and an accompanying table identifies the type of each word detected. The command decoder then interrogates these tables to obtain and check the data required for each command.

The input rules are:

(a) One or more blanks or a comma between words.

(b) The line is terminated by an * and anything following is a comment.

(c) Numerics start with Ø to 9, +, -, or . Anything else is an alphanumeric that is either a command or a matrix name.

(d) Numerics are integers unless a . or E is included, the E being used to enter exponents (e.g. 1E-6). Data items such as matrix sizes must be input as integers but matrix elements may be real or integer although they are stored and operated on only as reals.

(e) Repeated matrix elements may be input e.g. 1Ø(Ø).