WN-46
29Jul71

UNIVERSITY OF QUEENSLAND

Computer Centre

NEWSLETTER - SPECIAL EDITION

Date: 28 July 1971

Authorization: Director of the Computer Centre

NEW FORTRAN SYSTEM

On Monday 2 August, a new FORTRAN system will be 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 minimise 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 in itself.

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
      :
      :
   20 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 read operation.

The format of the statement is

```
READ (u,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),
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)
20  FORMAT (1X, 'A(',I1,')Δ=Δ', F8.2)
    .
    .
30  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 (5,20,PACKED) UNPK
20    FORMAT(5A1)
      PRINT 30,PACKED,UNPK
30    FORMAT('PACKED = ',A5,'UNPACKED TO ',5A2)
C
C     *****TO SELECT A CHARACTER FROM A WORD*****
      FLAGS='MBCRF'
      DECODE(5,10,FLAGS) BUSY
10    FORMAT(1X,A1,3X)
      PRINT 40,BUSY
40    FORMAT(/' THE BUSY FLAG IS ',A1//)
C
C     *****TO CHANGE THE SECOND CHARACTER OF "FLAGS" TO BLANK*****
      OFF=' '
      ENCODE(5,10,FLAGS)OFF
      DECODE(5,10,FLAGS)BUSY
      PRINT 40,BUSY
C
C     *****TO SET UP A RUN TIME FORMAT ARRAY*****
      NUMBER=9
      ENCODE(27,50,FMT) NUMBER
50    FORMAT('('' MOVING ARROW '',',I3,'X''↑'')')
      PRINT 60,FMT
60    FORMAT(' THE FORMAT IS ',6A5)
      PRINT FMT
      END

EXIT
↑C
```

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 an * 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)
.
.
. . .
.
.
. . .
.
.
END

SUBROUTINE TYPE (V1, $, V2, $)
.
.
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 return
END                         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.

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,ΔREALΔIS *****
ΔINTEGERΔISΔ***,ΔREALΔISΔ52.35
```

Note that G type formats should be used if there is any uncertainity 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 > 1.0 IN MAGNITUDE

ASIN OF ARG > 1.0 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 > 1.0 IN MAGNITUDE
ASIN OF ARG > 1.0 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 = "777777000000
```

(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 of this Newsletter.

(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 ⩽ |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 ⩽ 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 ⩽ 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 recommnnded 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
```