



3 4456 0275667 6

ORNL/TM-10770

# ornl

**OAK RIDGE  
NATIONAL  
LABORATORY**

**MARTIN MARIETTA**

## An Expeditious Technique for Keyboard Input of Information to FORTRAN Computer Programs

D. H. Smith  
D. E. Goeringer

OAK RIDGE NATIONAL LABORATORY

CENTRAL RESEARCH LIBRARY

CIRCULATION SECTION

4500N ROOM 175

**LIBRARY LOAN COPY**

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this  
report, send in name with report and  
the library will arrange a loan.

UCN-7969 13 9 77

OPERATED BY  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Road, Springfield, Virginia 22161  
NTIS price codes—Printed Copy: A03; Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

*Analytical Chemistry Division*

**AN EXPEDITIOUS TECHNIQUE FOR KEYBOARD INPUT OF  
INFORMATION TO FORTRAN COMPUTER PROGRAMS**

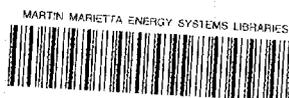
by

D. H. Smith and D. E. Goeringer

Date Published: May 1988

**NOTICE** This document contains information of a preliminary nature.  
It is subject to revision or correction and therefore does not represent a  
final report.

OAK RIDGE NATIONAL LABORATORY  
Oak Ridge, Tennessee 37831  
operated by  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
for the  
DEPARTMENT OF ENERGY  
Under Contract No. DE-AC05-84OR21400



3 4456 0275667 6



TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT . . . . .	1
INTRODUCTION . . . . .	3
DESCRIPTION . . . . .	4
General Program . . . . .	4
Subroutines . . . . .	9
CONCLUSIONS . . . . .	13
REFERENCES . . . . .	15
APPENDIX . . . . .	17



LIST OF TABLES

<u>Table No.</u>		<u>Page</u>
1	Hypothetical Prompt Screen . . . . .	5
2	Hypothetical Screen . . . . .	7
3	Hypothetical Screen . . . . .	8
4	Simple Cursor Control Subroutines . . . . .	10
5	Compound Cursor Subroutines . . . . .	11
6	Summary of VIDEO Function . . . . .	12
7	Summary of ERSLIN and ERSDIS Functions . . . . .	12



An Expeditious Technique for Keyboard Input of  
Information to FORTRAN Computer Programs

D. H. Smith and D. E. Goeringer

ABSTRACT

A technique for rapid and accurate input of information via a video display terminal to FORTRAN computer programs has been developed. The overall approach is described; it is applicable to any programming environment based on a similar system with only the most rudimentary control of graphics functions. The specific application in our laboratory, using DEC VT-100-compatible terminals, is also described and illustrated with examples. Listings of the graphics subroutines for use with a DEC RT-11 system are provided.



## INTRODUCTION

One of the more challenging jobs for the programmer is to make the final product as ergonomic, as "user-friendly", as possible. The input of the information necessary for the program to execute properly is probably the area most difficult to address. Instructions and prompts that are crystal clear to the programmer tend to be cryptic and misleading to the people who will be using the program. This user interface is a crucial ingredient to any commercially successful database or spread-sheet program, and it repays the efforts of even the non-professional programmer many times over to make his product as easy to use as possible.

The Analytical Spectroscopy Section of the Analytical Chemistry Division has had software to process mass spectrometric data in operation for over a decade.<sup>1-4</sup> These programs operate the mass spectrometers, acquire and process isotopic data, and write reports tabulating the results, most frequently in the form of the isotopic composition of the analyte element. The programs have, naturally, evolved with time; indeed, they are still doing so. Thousands of samples are analyzed every year with our five isotope ratio mass spectrometers. Many ORNL programs are supported by these instruments, including those related to reactor research and maintenance, environmental concerns, and isotope enrichment. Virtually any element can be a sample; some samples are analyzed via isotope dilution to give the concentration of the element; and some samples have interferences from isotopes of other elements that must be corrected. In other words, the programs are quite complex and, to retain their flexibility and generality, need a fair amount of information to execute as desired.

Technicians operate the instruments and enter the information necessary for program execution. While these people are talented individuals, they are not computer literate: they are unfamiliar with the internal operation of the computers and with the programs themselves. It is thus necessary to provide prompt messages that are meaningful to them as operators of mass spectrometers rather than the terser ones that would be satisfactory for the programmer. In addition, mistakes are occasionally made in entry of data. It is

essential that recovery from any conceivable error (and even some inconceivable ones) be as painless as possible. Editing programs have been written to allow correction of errors after the fact, but it is far more efficient if they can be corrected during the entry process; most errors are detected at that time.

We have developed a programmatic approach that in our opinion is exceptionally convenient and simple for the user. Instructions can be made as clear as desired, and error correction is simple, straightforward, and certain. Although our code was developed for a DEC RT-11-based system, the general approach is applicable to any computer system with keyboard-monitor input of data in any environment for any application. (It was actually first developed in BASIC by one of us at home on his personal computer for a program to catalog his record collection.) Only the most rudimentary functions are necessary; our subroutines only control cursor position and erase selected portions of the screen.

## DESCRIPTION

### General Program

The overall approach uses one screen of data as its basic input unit; by this we mean that, although there may be several screens of data to be input, only one at a time is entered, examined, corrected, and approved. It would be possible to have a final check of all input before proceeding to the main program, but we have not implemented it here.

Editing of input is predicated on the assumption that, with the appropriate graphics commands, one can move the cursor at will to any point on the monitor screen. This report is thus not applicable to systems lacking this ability or for those based on printing terminals.

After clearing the screen, the first display lists prompt messages vertically down its left side; we have found ten to fifteen to be a convenient number of variables with which to work for one screen of input. The messages can be of any length, but it is desirable to restrict them to the first 30 or 40 columns of an 80-column screen.

For messages longer than this offset, it is better to use two lines for the prompt than to extend it into the input region. Each prompt message is accompanied by an index code that is associated with it. We use numbers as index codes, but other schemes could be implemented if desired. Table 1 is an example of a hypothetical screen.

---

TABLE 1  
Hypothetical Prompt Screen

PERSONAL INFORMATION

1. Name
  2. Street address
  3. City
  4. State
  5. Zip code
  6. Social security number
  7. Date of birth
- 

Seven different parameters are called for under the general heading of "PERSONAL INFORMATION". It is highly desirable to have the entry for all parameters start in the same column. The mind and eye of the operator come to anticipate the location of the cursor, thus improving efficiency and accuracy of data input. By the same token, it is desirable for all modules of a large program to use a consistent format for input, maintaining, as closely as possible, the same starting columns, layouts, etc. between input screens. With the graphics subroutines described below, it is perfectly possible to have more than one entry per line, but it is visually confusing to do so. We often use column 30 as our initial entry column. This allows space for 25 or so characters for prompting messages and up to 50 characters for variables on one line.

During input, the program directs the cursor sequentially through all the variables displayed; in our example, this would be seven. Hitting the "Return" key signals the end of input for any given variable, at which the program jumps the cursor to the correct line and column for input of the next parameter. A number of underline characters, corresponding to the width of the entry field for that variable, is displayed beginning in the initial input column. For example, if a number between 0 and 999 is wanted, three underline characters are displayed beginning at the entry column. This is particularly helpful for long alphanumeric variables; without visual assistance, it is difficult to keep track of one's location in the input field if more than six or seven characters are called for. When values have been entered for all variables, a prompt appears at the bottom of the screen, telling the operator to enter a "0" (zero) or, for DEC computers, "Return" if all entries are correct. The index number corresponding to the incorrectly entered variable is entered otherwise. The operator can now verify his entries at his leisure. Upon entry of a non-zero value, the cursor is shifted to the place on the screen determined by the value of the index just entered. A new value for the parameter in question is entered. Upon entry of the next "Return" (the program can be written so that the value of the parameter need not be changed), the prompt appears again at the bottom of the screen asking for an index value for a correction. These steps are repeated until the operator signals all is correct, whereupon execution of the next section of the program commences. Our previous example is given in Table 2 with values entered for the various parameters and with the prompt message listed below.

TABLE 2  
Hypothetical Screen

## PERSONAL INFORMATION

1. Name	Malcolm Wellbeloved
2. Street address	3333 N. Insipid Drive
3. City	Unterwaltersdorf
4. State	NO
5. Zip code	45678
6. Social security number	314 15 9265
7. Date of birth	5-5-1787

0 if OK, index number otherwise \_

---

There are two places on the screen where the cursor may logically be positioned to await corrected input. The first is at the initial entry line and column for that variable. If this alternative is chosen, the original value must be erased from the screen before input of a new value is started. The second location is on the same line as the original entry but offset to the right from it. This allows the original value to remain displayed on the screen while a new one is entered. The disadvantage of this second method is that one is restricted as to the length of the variables that can be conveniently handled. If, for example, one wants an alphanumeric variable 30 characters in length (not overly generous), one would not be able to keep everything on one line (25 characters for a prompt message and 30 each for the original and new values for the parameter; 80 is the maximum number of characters per line). We have, therefore, usually chosen the first of the alternatives just described. For the sake of visual clarity, however, we have chosen to use the second alternative in our examples below.

To illustrate the positioning of the cursor preparatory to input of a corrected value, consider the data in Table 2. If Mr. Wellbeloved lived in Nebraska rather than Niederoesterreich, a 4 would be entered. The cursor would be moved to a position between the end of the original input field and the right side of the screen--there must be enough room on the line to hold the entire input field for the corrected entry. In this case, it would be positioned at the end of the line labelled "4. State", and a new value for this variable would be entered. Upon receiving the "Return" keystroke, the original and corrected values are both erased, and the corrected value inserted into the position of the original input field.

---

TABLE 3

Hypothetical Screen

PERSONAL INFORMATION

1. Name	Malcolm Wellbeloved
2. Street address	3333 N. Insipid Drive
3. City	Unterwaltersdorf
4. State	<u>NO</u>
5. Zip code	45678
6. Social security number	314 15 9265
7. Date of birth	5-5-1787

0 if OK, index number otherwise 4

---

In Table 3, a new entry for the state is being called for. Two underline characters indicate that the two-character postal service abbreviation is wanted rather than the full name. After entering the correct value in the field shown, the operator reviews all input, makes any needed corrections, and, when all is correct, enters "0" (or, on DEC equipment, enters "Return" directly) to proceed to the next section of the program.

The ability to review a block of input and correct any erroneous entries at will is one the most attractive features of our method. It has resulted in far fewer input errors being transmitted to the final calculational programs than had been the case for our previous methods of data entry.

### Subroutines

Movement of the cursor is controlled by a number of subroutines. These were developed in DEC FORTRAN IV. They would execute more rapidly if they had been written in assembly language, but in practice the movement of the cursor is fast enough to be completely insignificant on a human time scale; the cursor moves between any two locations on the screen in less time than it takes to enter a single character at the keyboard, giving the subjective impression of instantaneous movement. There thus seems to be no reason to go to the trouble to develop assembly code for the routines. The FORTRAN code was implemented by executing the appropriate escape codes to achieve the desired result; they apply to DEC VT-100-compatible terminals.

It should be pointed out that some care must be exercised by the programmer to keep track of the current cursor position. Techniques that work perfectly under other conditions cause loss of cursor calibration in this graphics mode. For example, PUTSTR, the FORTRAN routine to output a string, automatically includes a line feed at the end, causing the cursor to be positioned one line too low. We have not found a way to circumvent this behavior using PUTSTR. One could, of course, go to the trouble of taking this into account, but it seems more straightforward to use the WRITE statement and output the string using A fields. It is easy, however, to cause the cursor to be one line too low using WRITE statements as well. For example, if one wants to output a two-character string, one must use the following format statement to retain cursor calibration:

```
FORMAT(1H+,A2,$)
```

Both the 1H+ and the \$ are necessary to prevent errors in cursor location. If 3A2, for example, is substituted for A2, the cursor will again stubbornly position itself one line too low. It is not clear to us why this should be. The graphics commands refer to a specific screen location, but the operating system seems to lose track of just where the cursor is.

Table 4 lists the names of the routines and short descriptions of their functions.

---

TABLE 4	
Simple Cursor Control Subroutines	
<u>Name</u>	<u>Description</u>
CLRSCN	Clears entire screen.
HOMCUR	Homes cursor.
MOVCUR	Moves cursor specified number of lines and columns from the present location.
CURPOS	Positions cursor at specified line and column.
CURFND	Reports cursor location.
SETSCL	Sets a scrolling region.
VIDEO	Selects between normal/reverse, bold/nobold, blink/noblink and underline/nounderline video modes.
ERSLIN	Erase specified parts of the line containing the cursor.
ERSDIS	Erase specified parts of the display.
PROMPT	Prints specified number of underline characters for the input field.

---

A number of handy routines has been developed that combine two or more of the simple ones. Some of the more useful of these are listed in Table 5.

TABLE 5

## Compound Cursor Subroutines

<u>Name</u>	<u>Description</u>
CLRHOM	Clears screen and homes cursor (CLRSCN + HOMCUR).
CLR24	Clears lines 20-24 (ERSLIN in a loop).
CURLOC	Positions cursor and displays a specified number of underlines (CURPOS + PROMPT).
CURCAL	Recalibrates cursor position; sometimes necessary when shifting screens.

---

Although many of the short descriptions in Tables 4 and 5 are enough to define fully the associated functions, some require amplification. MOVCUR has two arguments, NCOLS and LINES. Using the current cursor location as a reference, it moves the cursor the specified number of columns and lines. Positive numbers move the cursor to the right and down; negative values move it left and up. Any combination of positive and negative numbers within the ranges of lines and columns of the display is allowed. CURPOS also has two arguments, NCOL and NLINE, which designate the screen coordinates to which the cursor should move; the range for NLINE is 1-24 and for NCOL 1-80. CURFND has the same two arguments as CURPOS. It, however, reports the present position of the cursor. It is especially useful when used in conjunction with MOVCUR. SETSCL has two arguments, LINTOP and LINBOT. These arguments specify the line numbers for the top and bottom, respectively, of the scrolling area of the screen. This useful function allows one to save a display in one area of the screen while another scrolls as needed. This routine homes the cursor. VIDEO sets the video mode and uses a single argument, MODE. MODE is a bitwise variable that controls the display mode. Table 6 defines the values for MODE.

TABLE 6  
Summary of VIDEO Function

<u>Bit</u>	<u>Decimal</u>	<u>Clear (0)</u>	<u>Set (1)</u>
0	1	normal video	reverse video
1	2	bold off	bold on
2	4	blink off	blink on
3	8	underline off	underline on

---

The value for MODE is obtained by adding the decimal values that correspond each bit that is "set". The programmer selects which options are to be activated and finds the appropriate sum. Thus one would use 9 as an argument to have reverse video with underline on.

ERSLIN has one argument that defines the direction and extent of erasure on the line containing the cursor. It is defined in Table 7. ERSDIS performs a function similar to ERSLIN, but for the whole screen rather than for just one line. It, too, takes one argument whose operation is described in Table 7.

---

TABLE 7  
Summary of ERSLIN and ERSDIS Functions

<u>ERSLIN</u>	
<u>Value</u>	<u>Effect</u>
1	Erases entire line containing cursor.
2	Erases from cursor to end of line, including cursor position.
3	Erases from start of the line through cursor position.
<u>ERSDIS</u>	
1	Erases entire screen.
2	Erases from cursor to end of screen, including cursor position.
3	Erases from start of screen through cursor position.

---

PROMPT takes one argument that indicates the number of underline characters to print to the screen. It is up to the programmer to have the cursor correctly positioned before issuing the call to the subroutine. This is, of course, true for many of these graphics routines.

Of the compound functions, only CURCAL seems to require further discussion. This routine calls CLRHOM (itself compounded of CLRSCN and HOMCUR) and writes a do-nothing line to the screen. This line is formatted as (LH+,\$). It is not clear to the authors why this routine is needed. Its function is to recalibrate the position of the cursor. Its need arises when the computer loses track of the cursor position. This usually occurs when one screen of data has been successfully entered, the screen cleared, the cursor homed, and a new series of prompts displayed. The computer will often position the cursor one line too low, an aggravating condition that renders input of information more challenging than it should be. CURCAL corrects this situation.

#### CONCLUSIONS

The approach embodied in this report has been in operation for well over a year. It has been well received by all who use the programs into which it is integrated; operators range from daily users to occasional. It seems easier for a person to remember how to make input correctly when this method is used than it was for earlier methods.

Although this manual is intended primarily as a reference for the authors and others in their section, it should be useful to any programmer who wants to incorporate an efficient, accurate method of manual keyboard input to a computer program. The listings for the subroutines, which are all a page or less long, are included in the appendix. Any reader who wants more information is invited to get in touch with either of the authors.



#### REFERENCES

1. D. H. Smith, Chem, Biomed, and Environ Instrum 10, 27 (1980).
2. D. H. Smith, H. S. McKown, W, H. Christie, R. L. Walker, and J. A. Carter, USERDA Report ORNL/TM-5485, June 1976.
3. D. H. Smith, USDOE Report ORNL/TM-7002, September 1979.
4. D. H. Smith, USDOE Report ORNL/TM-8356, August 1982.



APPENDIX  
LISTINGS OF GRAPHICS SUBROUTINES





C  
C  
C  
C  
C

SUBROUTINE: CLRSCN

PURPOSE: To clear the screen

```

subroutine clrscn
logical*1 vtclr(5)
data vtclr/'33','L','2','J','200/

call print(vtclr)
return
end

```

C  
C  
C  
C

SUBROUTINE CLRHOM

PURPOSE: To clear screen and home cursor.

```

Subroutine Clrhom
Call Clrscn
Call Homecur
return
end

```

C  
C  
C  
C

SUBROUTINE: CLR04

PURPOSE: To clear lines 20-24 for messages.

```

Subroutine Clr04
call curpos(1,20)
call eredis(2)
return
end

```

C  
C  
C  
C  
C  
C  
C

SUBROUTINE: CURCAL

PURPOSE: To set cursor correctly located; this is required for no apparent reason on some occasions, especially those where the program is reentered after one pass through the calculations.

```

Subroutine Curcal
call clrhom
write(7,9946)
return

```

C  
9946

```

format(1H+,$)
end

```

C  
C  
C  
C  
C  
C  
C  
C

SUBROUTINE: CURFND(NCOL,NLINE)

PURPOSE: To report location of cursor

ARGUMENTS:

ncol - current column #  
nline - current line #

```

subroutine curfnd(ncol,nline)
logical*1 vstrn(3),hstrn(4),reapos(5),postrn(10),
+ scacon(3),scacof(3)
integer spcm0d
data reapos/'33','L','6','n','200',spcm0d/'10000',jsw/'44',
+ scacon/'35','S','200',scacof/'35','T','200/

jswold=ipeek(jsw)
jswspc=jswold.or.spcm0d

call print(reapos)
call ipeek(jsw,jswspc)
call print(scacon)

j=0
100 j=j+1
inchr=ittirr()
postrn(j)=inchr
if(postrn(j).ne.'R')goto 100
nend=j
j=j+1
postrn(j)='0'
call ipeek(jsw,jswold)
call print(scacof)

nsemi=index(postrn,';',3)
call substr(postrn,vstrn,3;nsemi-3)
call substr(postrn,hstrn,nsemi+1;nend-nsemi-1)

decode(nsemi-3,1000,vstrn)nline
decode(nend-nsemi-1,1000,hstrn)ncol
1000 format(i3)

return

end

```

```

C      SUBROUTINE: CURLOC
C
C      PURPOSE: To locate cursor at desired position and underline
C                indicated number of spaces for input field.
C                Reverse video is activated; this means it should be
C                deactivated upon leaving this subroutine.
C
C      ARGUMENTS:
C                ncol - column no.
C                nline - line no.
C                nblank - no. of blanks
C
C      Subroutine Curloc(ncol,nline,nblank)
C      Call Curpos(ncol,nline)
C      Call Promt(nblank)
C      call Video(1)
C      return
C
C      end

```

```

C
C      SUBROUTINE: CURPOS(NCOL,NLINE)
C
C      PURPOSE: To position cursor at selected column and line
C
C      ARGUMENTS:
C                ncol - column number (must be 1-80)
C                nline - line number ( must be 1-25)
C
C      subroutine curpos(ncol,nline)
C      logical*1 esi(3),cur(2),colstr(3),linstr(3),semi(2)
C      data esi/'33','L','200';cur/'H','200';semi/'f','200/'
C
C      encode(3,1000,colstr)ncol
1000  format(i2)
C      colstr(3)='200'
C      if(colstr(1).ne.'40')goto 50
C      colstr(1)=colstr(2)
C      colstr(2)='200'
50    encode(3,1000,linstr)nline
C      linstr(3)='200'
C      if(linstr(1).ne.'40')goto 100
C      linstr(1)=linstr(2)
C      linstr(2)='200'
100   call print(esi)
C      call print(linstr)
C      call print(semi)
C      call print(colstr)
C      call print(cur)
C
C      return
C      end

```

C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C

SUBROUTINE: ERSDIS(MODE)

PURPOSE: To erase selected parts of the display

ARGUMENT:

mode - type of display erase to perform  
 1 = entire screen  
 2 = from cursor to end of screen (incl cursor)  
 3 = from beginning of screen to cursor  
 (incl cursor)

```

subroutine ersdis(mode)
logical*1 eras(5),eos(5),bos(5)
data eras/'33','[','2','J','200',eos/'33','[','0','J','200',
+bos/'33','[','1','J','200/

if(mode.eq.1)call print(eras)
if(mode.eq.2)call print(eos)
if(mode.eq.3)call print(bos)
return

end

```

C  
C  
C  
C  
C  
C  
C  
C  
C  
C

SUBROUTINE: ERSLIN(MODE)

PURPOSE: To erase line or segments of line

ARGUMENT:

mode - type of erase to perform  
 1 = entire line containing cursor  
 2 = from cursor to end of line (incl cursor)  
 3 = from start of line to cursor (incl cursor)

```

subroutine erslin(mode)
logical*1 eras(5),eol(5),bol(5)
data eras/'33','[','2','K','200',eol/'33','[','0','K','200',
+bol/'33','[','1','K','200/

if(mode.eq.1)call print(eras)
if(mode.eq.2)call print(eol)
if(mode.eq.3)call print(bol)
return

end

```

C  
C  
C  
C  
C

SUBROUTINE: HOMCUR

PURPOSE: To home the cursor

```
subroutine homcur
logical*1 home(7)
data home/'33','C','1','1','1','1','H','200/

call print(home)
return
end
```

```

C
C      SUBROUTINE: MOVCUR(NCOLS,NLINES)
C
C      PURPOSE: To move cursor a variable # columns and lines
C
C      ARGUMENTS:
C          ncols - number of columns to move
C                  positive = move right
C                  negative = move left
C          nlines - number of lines to move
C                  positive = move down
C                  negative = move up
C
C
C      subroutine movcur(ncols,nlines)
C      logical*1 curup(2),curdn(2),curfwd(2),curbwd(2),csi(3),
C      +colstr(4),linstr(4)
C      data curup/'A','200',curdn/'B','200',curfwd/'C','200',
C      +curbwd/'D','200',csi/'33','L','200/
C
C      if(ncols.ge.0)goto 200
C      encode (3,1000,colstr)ncols
1000  format(i3)
C      i=1
C      do 50 J=1,3
C      if(colstr(J).eq.'40 .or. colstr(J).eq.'-')goto 50
C      colstr(i)=colstr(J)
C      i=i+1
50    continue
C      colstr(i)="200
C      call print(csi)
C      if(ncols.lt.0)goto 100
C      call print(colstr)
C      call print(curfwd)
C      goto 200
100   call print(colstr)
C      call print(curbwd)
C
C      if(nlines.ge.0)goto 400
C      encode (3,1000,linstr)nlines
C      i=1
C      do 250 J=1,3
C      if(linstr(J).eq.'40 .or. linstr(J).eq.'-')goto 250
C      linstr(i)=linstr(J)
C      i=i+1
250  continue
C      linstr(i)="200
C      call print(csi)
C      if(nlines.lt.0)goto 300
C      call print(linstr)
C      call print(curdn)
C      goto 400
300  call print(linstr)
C      call print(curup)
C
C      return
C      end

```

```
C
C      SUBROUTINE: PROMPT(IBLANK)
C
C      PURPOSE: To designate an input field using underline
C              from FORTRAN.
C
C      ARGUMENT:
C              iblank - # blanks to underscore.
C
      subroutine prompt(iblank)
      logical*1 space(2)
      data space/'40','200/'

      call video(8)
      do 50 i=1,iblank
      call print(space)
50    continue
      call movcur(-iblank,0)
      call video(0)

      return
      end
```

C  
C  
C  
C  
C  
C  
C  
C  
C  
C

SUBROUTINE: SETSCL(LINTOP,LINBOT)

PURPOSE: To define scrolling region

ARGUMENTS:

    lintop - line # for first line in scrolling region  
    linbot - line # for last line in scrolling region

NOTE: This subr also homes the cursor

```

subroutine setscl(lintop,linbot)
logical*1 csi(3),semi(2),topbot(2),topstr(3),botstr(3),setpbm(2)
data csi/'33','F','200',semi/'i','200',setpbm/'r','200/

1000 encode(3,1000,topstr)lintop
format(i2)
topstr(3)='200
if(topstr(1).ne.'40)goto 50
topstr(1)=topstr(2)
topstr(2)='200
50 encode(3,1000,botstr)linbot
botstr(3)='200
if(botstr(1).ne.'40)goto 100
botstr(1)=botstr(2)
botstr(2)='200
100 call print(csi)
call print(topstr)
call print(semi)
call print(botstr)
call print(setpbm)

return
end

```

C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C

SUBROUTINE: VIDEO(MODE)

PURPOSE: To set output to reverse or normal video

ARGUMENT:

mode - video output mode  
 bit 0 = 0 : normal video  
 bit 0 = 1 : reverse video  
 bit 1 = 0 : bold off  
 bit 1 = 1 : bold on  
 bit 2 = 0 : blink off  
 bit 2 = 1 : blink on  
 bit 3 = 0 : underline off  
 bit 3 = 1 : underline on

Note: a negative value for mode resets  
 all attributes.

```

subroutine video(mode)
logical*1 alloff(5),rvideo(5),nvideo(6),boldon(5),boldof(6),
+blnkon(5),blnkof(6),undlon(5),undlof(6)
data alloff/'33,'[','0','m','200/,
+trvideo/'33,'[','7','m','200/,
+trvideo/'33,'[','2','7','m','200/,
+tboldon/'33,'[','1','m','200/,
+tboldof/'33,'[','2','2','m','200/,
+tblnkon/'33,'[','5','m','200/,
+tblnkof/'33,'[','2','5','m','200/,
+tundlon/'33,'[','4','m','200/,
+tundlof/'33,'[','2','4','m','200/

if(mode.lt.0)goto 500
itest=mode.and.'1
if(itest.eq.'1)call print(rvideo)
if(itest.eq.'0)call print(nvideo)
itest=mode.and.'2
if(itest.eq.'2)call print(boldon)
if(itest.eq.'0)call print(boldof)
itest=mode.and.'4
if(itest.eq.'4)call print(blnkon)
if(itest.eq.'0)call print(blnkof)
itest=mode.and.'10
if(itest.eq.'10)call print(undlon)
if(itest.eq.'0)call print(undlof)
return

500 call print(alloff)
return

end

```

Internal Distribution

1. K. G. Asano
2. J. A. Carter
3. W. H. Christie
- 4-8. D. E. Goeringer
9. B. C. Grant
10. J. M. Keller
11. L. N. Klatt
12. H. S. McKown
13. W. D. Shults
- 14-18. D. H. Smith
19. P. J. Todd
20. R. E. Valiga
21. R. L. Walker
22. A. Zucker
23. Central Research Library
24. Document Reference Section
25. ORNL Patent Office
- 26-27. Laboratory Records Department
28. Laboratory Records-RC

External Distribution

29. Assistant Manager for Energy Research and Development,  
Department of Energy, Oak Ridge Operations, Oak Ridge, TN 37831
- 30-39. Office of Scientific and Technical Information, Oak Ridge, TN  
37831

