Fortran

programming language for the Commodore 64

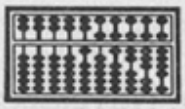
Abacus

Fortran-64

Programming language for the Commodore 64

Published By:

Abacus



Copyright Notice

Warranty

Abacus makes no warranties, expressed or implied, as to the fitness of this software package for any particular purpose. In no event will Abacus be liable for consequential damages. Abacus will replace any copy of this software which is unreadable, if returned within 30 days of purchase. Thereafter, there will be a nominal charge for replacement.

First Printing, November 1988
Printed in U.S.A.

ISBN 0-916439-91-7

TABLE OF CONTENTS

WRITING A FORTRAN PROGRAM

Writing a Fortran file involves a number of different files. Before starting your programming effort, you must create a data disk (see Creating a Data Disk). There are a number of files on your disk that are vital to the programming process. The steps needed to produce an executable program follow (see page 5, Program Creation).

The first file you will create is the Fortran source file (the file that holds your Fortran statements). The program must adhere to the format described on pages 4-6, Program Format. If the program is to be executed it must contain one main program unit and any number of subroutine or function units. If the file does not contain a main program it can still be compiled and later linked with a file that does have a main program.

The source file can be created using two methods. The first is to use any word processor which produces sequential ASCII output files.

The other alternative, is to create a file using the BASIC editor (see Back to BASIC) and translate it into a sequential file using the Translator (see Translator mode). The TEST program on page 8 gives an example of a Fortran program written in the BASIC editor, before it is translated. Each line is entered as in BASIC, with the following exception: A colon (:) must appear on the line between the line number and the statement. After the program is written and all editing is completed, it is simply saved to the disk using the BASIC SAVE command.

Once a sequential file has been created with either method, enter the Compiler mode to compile the source program into an object file. The object file contains relocatable, binary machine code.

At this point object files can be combined by linking them together and loading them into memory. This is done in the Linker mode.

After the linking process is completed, an executable program resides in memory. If the BASIC LIST command is used, a REM statement appears on the first line along with the name of your

Fortran program. A SYS statement also exists with the execution address.

The rest of the program is in machine language. At this point the program can be saved to disk or it can be executed using the BASIC RUN command. This program can be put on any disk. It does not have to reside on your data disk.

COMPILING A PROGRAM

The Fortran-64 disk contains many example programs. The following is an example of the complete compilation process:

1) Load Fortran-64 (see GETTING STARTED.)

2) Press the F6 key for translator mode (see Translator mode.)

2) Enter DOC-READER.FOR as the filename then press Return.

3) Enter DOC-READER.SEQ as the sequential filename then press Return. The file will be converted to a sequential file. Press Return when the translation process is finished.

4) Press F1 to enter Compiler mode (see Compiler mode.) Enter DOC-READER.SEQ as the source filename then press Return.

5) Enter DOC-READER.OBJ as the object filename then press Return. Enter n for HARDCOPY. The file will be displayed on the screen as it is compiled. Press the Return key when the compilation process is finished.

6) Press F3 to enter the Linker mode (see Linker mode). Press y to confirm entering Linker mode. Enter DOC-READER.OBJ as the object filename and press Return.

7) Enter n in response to the LIST MODULES questions.

8) Enter STRING.LIB in response to the UNDEFINED EXTERNAL message.

9) The program is ready to save and run. To save the program to disk, enter SAVE "DOC-READER",8 and press Return .

10) Enter RUN and press Return to run the program. Enter 0 for no printout. Next enter TOPICS.DOC. The TOPICS.DOC will be displayed. Read each screen then press the Return key to move to the next screen.

THE PROGRAM FORMAT

```
      PROGRAM name
        declarations
              °
              °
label statements
              °
              °
      END
*
*  COMMENTS
*
      SUBROUTINE name (arguments)
        declarations
              °
              °
      RETURN
      END
*
*
      FUNCTION name (arguments)
        declarations
              °
              °
        name *??
        RETURN
        END
```

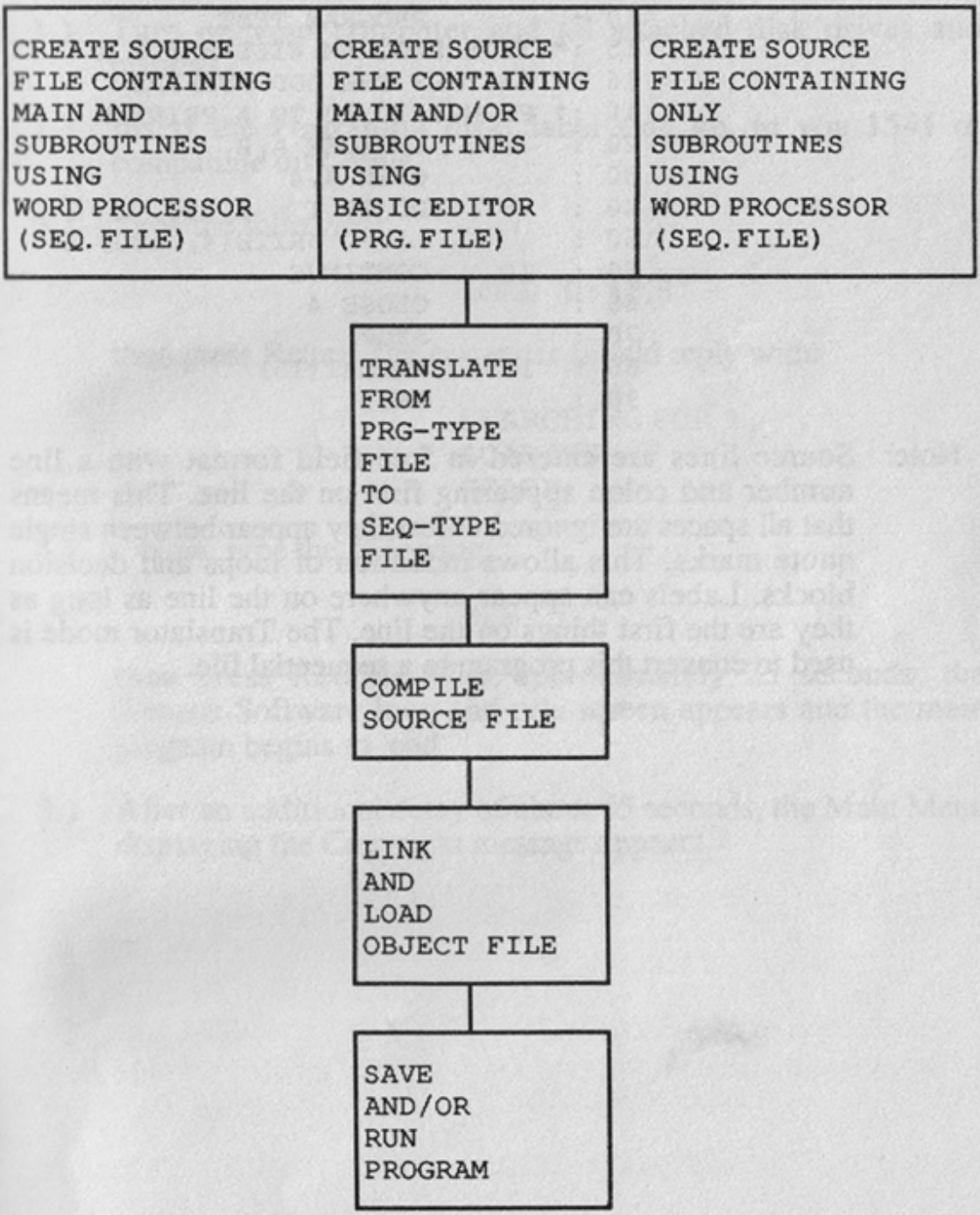
Note: Source lines are entered in free field format. This means that all spaces are ignored unless they appear between single quote marks. This allows indention of loops and decision blocks. Labels can appear anywhere on the line if they are the first on the line.

Comment lines must start with *. Blank lines are ignored.

Any statement can be continued onto the next line by placing a # at the end of a line. The following example would be considered one line although it is typed in on two:

```
WRITE(4,*) I,J,A,B,C,D#,X,Y,Z
```

PROGRAM CREATION



TEST PROGRAM (TRANSLATABLE)

```
10 :         PROGRAM TEST
15 :* THIS PROGRAM WILL WRITE A
16 :* LIST OF THE ODD NUMBERS
16 :* FROM 1 TO 10 TO A PRINTER
20 :         INTEGER A,B
30 :         OPEN 4,4
40 :         DO 10 I = 1, 10, 2
50 :               WRITE(4, 100) I
60 :  10     CONTINUE
65 :         CLOSE 4
70 :         STOP
80 :  100    FORMAT(I5)
90 :         END
```

Note: Source lines are entered in free field format with a line number and colon appearing first on the line. This means that all spaces are ignored unless they appear between single quote marks. This allows indention of loops and decision blocks. Labels can appear anywhere on the line as long as they are the first things on the line. The Translator mode is used to convert this program to a sequential file.

GETTING STARTED

1.) Turn on your computer and all attached disk drives and printers.

2.) Insert the Fortran-64 disk, label side up, in you 1541 or compatible disk drive.

3.) Type the following:

```
LOAD "0:*",8
```

then press Return.The computer should reply with:

```
SEARCHING FOR *
LOADING
READY
```

4.) Now, type the following:

```
RUN
```

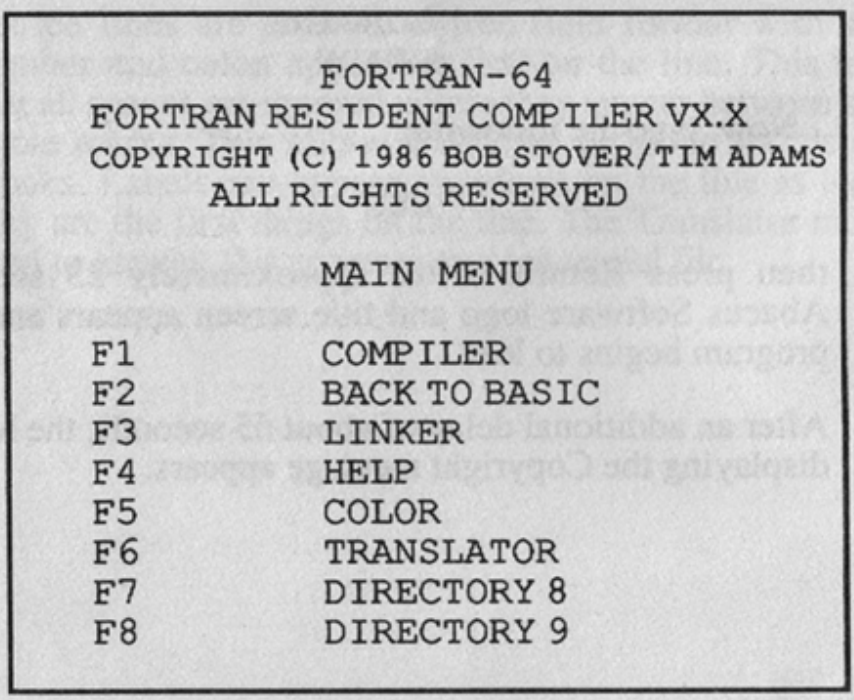
then press Return. After approximately 25 seconds, the Abacus Software logo and title screen appears and the main program begins to load.

5.) After an additional delay of about 65 seconds, the Main Menu displaying the Copyright message appears.

THE MAIN MENU

The Main Menu lets you select the Compiler, Linker and other features by pressing a function key. When depressed, the function keys perform the following:

F1 Enter the Compiler mode
F2 Back to BASIC
F3 Enter the Linker mode
F4 Enter the Help mode
F5 Enter the Color mode
F6 Enter the Translator mode
F7 View the directory on disk drive #8
F8 View the directory on disk drive #9



Screen for Main Menu

COMPILER MODE

If you press function key F1 you enter Compiler mode. You are asked to type the name of your source file.

```
ENTER SOURCE FILENAME:
```

Type the name of the source file to be compiled. You can specify a separate disk drive by typing as follows:

To select drive #8:

```
filename or D8:filename
```

To select drive #9:

```
D9:filename
```

The filename typed is a string of 1 to 16 characters.

Note: Only drives #8 and #9 are supported in the Compiler mode. However, the syntax for using these two drives permits maximum flexibility, by allowing the user to enter the source file from either drive and direct the object file to either drive.

Note: After entering the Compiler mode, function key F1 can be used to return to the Main Menu, before the object filename is specified.

After the source filename is entered, the compiler asks for the object filename as follows:

```
ENTER OBJECT FILENAME:
```

The object filename can be entered using the same syntax as described for entering the source filename.

Note: If the object filename already exists on the disk, an error message appears. A new filename can then be entered or the old file can be replaced using the following syntax:

```
@0:filename or
D8:@0:filename or
D9:@0:filename
```

The compiler now asks if a printer listing of the source file during compilation is desired as follows:

```
HARDCOPY? (Y/N CR-N)
```

If a Y is entered, the compiler asks for the device number as follows:

```
HARDCOPY DEVICE? (4/5 CR-4)
```

If a N or CR is entered, the compiler begins to compile the source file entered and creates an object file on disk.

Any errors found during compilation are indicated either on the screen or on the hardcopy device (if selected). See the Error Description section of this manual for details on error determination.

Anytime during the compilation process, the compiler can be paused or aborted with the following keystrokes:

Run/Stop key	forces the compiler to pause
space bar	permits the compiler to continue after pause
Q	aborts the compilation process

After compilation, the results of the process are indicated as follows:

```
*****     ERRORS DETECTED.
@@@@@     SOURCE LINES READ.
```

where ***** is the number of errors detected and @@@@@ is the number of source lines encountered.

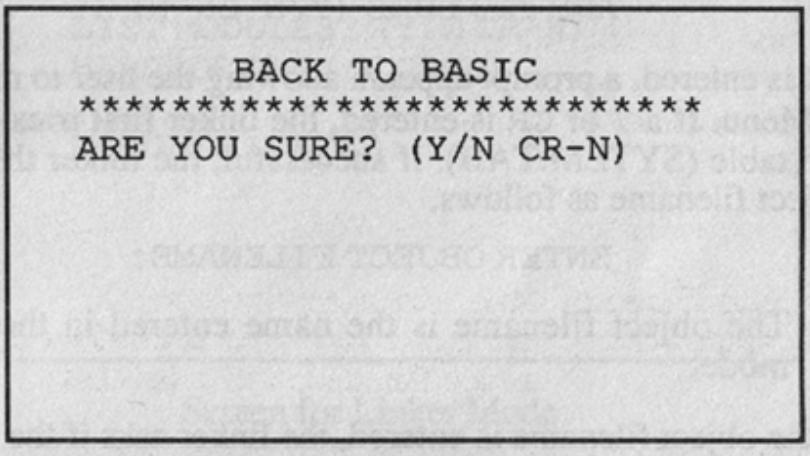
A prompt then appears allowing the user to return to the Main Menu.

BACK TO BASIC

After function key F2 is depressed, the user is given the option to return to BASIC (see Figure 2). This permits creation or editing of a Fortran source file while in BASIC. However, the following prompt allows the user to change his mind:

```
ARE YOU SURE? (Y/N CR-N)
```

Note: In order to allow creation or editing of your Fortran file with the BASIC editor and allow re-entry into Fortran-64 without reloading, a SYS 49152 must be performed immediately after returning to BASIC. After saving the file, a second SYS 49152 can be performed to return to the Main Menu.



Screen for Back to BASIC

LINKER MODE

After function key F3 is depressed, the Linker mode is entered. (See Figure 3)

Note: If linking is carried out, no further returns to the Main Menu are possible. However, if linking is performed, additional object files can be linked or relinked by typing `SYS 49152`, followed by the Return key.

Note: Due to memory limitations, only drive #8 is supported in Linker mode.

The first question asked by the linker is as follows:

```
ARE YOU SURE (Y/N CR-N)
```

If an `N` is entered, a prompt appears allowing the user to return to the Main Menu. If a `Y` or `CR` is entered, the linker first tries to load the system table (SYTEM.TAB). If successful, the linker then asks for the object filename as follows.

```
ENTER OBJECT FILENAME:
```

Note: The object filename is the name entered in the Compiler mode.

After the object filename is entered, the linker asks if the user wants a list of the modules loaded as follows:

```
LIST MODULES/ (Y/N CR-N)
```

If an `N` is entered, the linking process begins. If a `Y` or `CR` is entered, the linker asks if a printer listing of the loaded modules is desired as follows:

```
HARDCOPY? (Y/N CR-N)
```

Depending on the replay, all modules loaded during the linking process are listed on the printer or the screen.

Note: Due to memory limitations, only print device #4 is supported in Linker mode.

If an external subroutine is called in the main routine of a Fortran program but the external subroutine was compiled separately, a message is displayed as follows:

```
UNDEFINED EXTERNAL
```

```
                LINKER
*****************************

ARE YOU SURE? (Y/N CR-Y)
ENTER OBJECT FILENAME:
LIST MODULES? (Y/N CR-N)
HARDCOPY? (Y/N CR-N)

MODULES LOADED:
```

Screen for Linker Mode

The linker also displays the name of the subroutine and asks for the filename of the library where the user may have included this external subroutine as follows:

```
ENTER LIBRARY NAME:
```

After the filename is entered, the linking process continues. When linking is completed, the following message appears:

```
RUNABLE CODE RESIDES
FROM $0850 TO $****
```

Where $**** is the ending address.

Note: If a library filename is entered that doesn't contain the external subroutine requested or if any disk errors occur, an error message is displayed on the screen and a retry message appears. This can be helpful if libraries or modules are located on separate disks. This message allows the user to swap disks until the required file is located.

The final message to appear is the completed message as follows:

```
LINK COMPLETE
```

At this point, a runable/saveable version of the program is resident in memory. USe the normal save command to save the program to diskette.

HELP MODE

After function key F4 is depressed, the Help mode is entered. This mode describes the options available from the Main Menu.

A prompt appears allowing the user to return to the Main Menu.

```
                HELP
**********************************
F1      SELECTS COMPILER
        SYNTAX:
        FOR SOURCE OR OBJECT FILENAME:
        FILENAME OR D8:FILENAME
        OR D9:FILENAME:
F2      RETURNS TO BASIC:
F3      SELECTS LINKER:
        IF SELECTED:
ALLOWS FOR SAVE AND/OR RUN:
F4      SELECTS THIS HELP FEATURE:
F5      SELECTS COLOR CHANGE MENU:
F6      SELECTS TRANSLATOR:
F7      SELECTS DIRECTORY ON DRIVE 8:
F8      SELECTS DIRECTORY ON DRIVE 9:

PRESS RETURN WHEN READY:
```

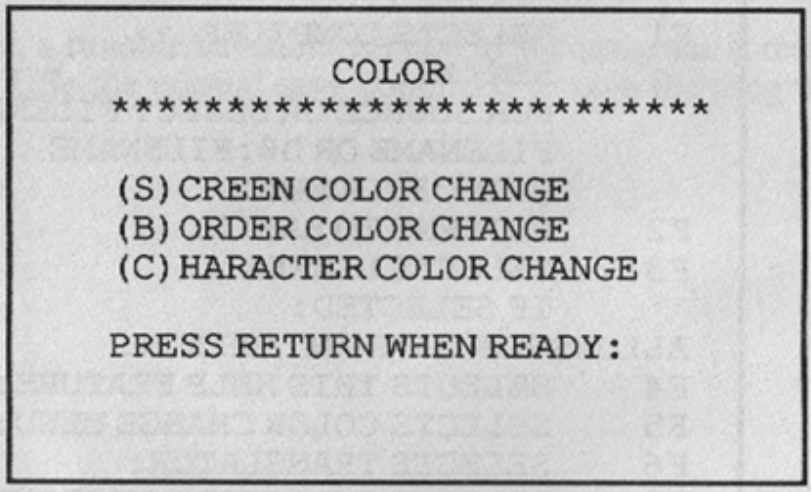
Screen for Help Mode

COLOR MODE

After function key F5 is depressed, the Color mode is entered. This mode allows the user to select the color you desire to work with.

A prompt appears allowing the user to return to the Main Menu.

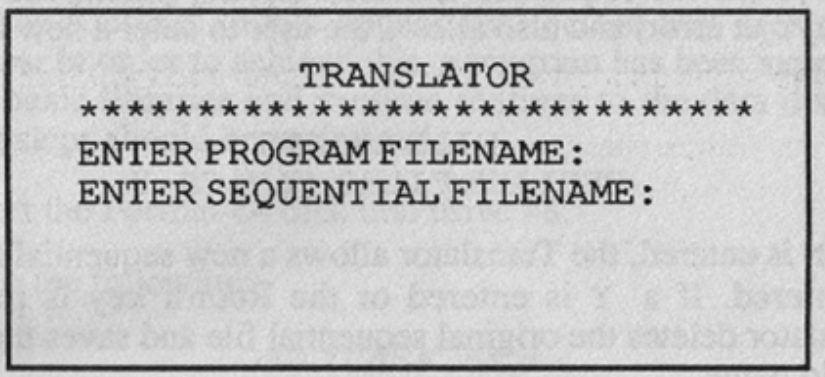
After the source filename is entered, the compiler asks for the object filename as follows:



Screen for Color Mode

TRANSLATOR MODE

After function key F6 is depressed, the Translator mode is entered.



Screen for Translator Mode

Note: Due to memory limitations, only drive #8 is supported in Translator mode.

The Translator first asks for the program filename as follows:

ENTER PROGRAM FILENAME:

The program filename is the name of the file originally created using the Commodore BASIC editor and saved on the disk.

The Translator then asks for the sequential filename as follows:

ENTER SEQUENTIAL FILENAME:

Note: The sequential filename is the name of the source file used as input to the Compiler in Compiler mode.

After entering this name the Translator converts your program file to a source file.

If a disk error is detected, it is displayed on the screen as follows:

DISK ERROR xx FOUND

where xx is the error number. (see your disk drive manual for error explanations)

However, if the sequential filename already exists, the Translator displays an error, and also allows the user to enter a new filename as follows:

```
FILE EXISTS.
REPLACE FILE? (Y/N CR-Y)
```

If an N is entered, the Translator allows a new sequential filename to be entered. If a Y is entered or the Return key is pressed, the Translator deletes the original sequential file and saves the file under the old name.

If a colon is not detected on a line, an error is also generated as follows:

```
ERROR: COLON MISSING ON LINE xx
```

Note: The sequential file is still generated after detecting missing colons. However, the program file must be corrected and retranslated before proceeding to the Compiler mode.

After successful completion of the Translation process, a message appears as follows:

```
TRANSLATION COMPLETE
```

A prompt then appears allowing the user to return to the Main Menu.

DIRECTORY

Function keys F7 and F8 allow the disk directory from drive #8 or #9 respectively, to be viewed. Pressing the spacebar while viewing the directory will pause the display. Depressing any key allows continued viewing. If the Run/Stop key is pressed or when the end of the directory is reached, a prompt appears allowing the user to return to the Main Menu.

CREATING A DATA DISK

In order to link a compiled program in the shortest amount of time, it is advisable to create a data or source disk. On this disk, the library and runtime routines should appear first before any user created source files. In order to achieve this, a program has been supplied to copy the basic libraries and runtime routines to the data disk. The following steps should be performed:

1.) Insert the Fortran-64 disk into drive #8.

2.) Type the following:

```
LOAD "CREATE",8
```

followed by a carriage return. The computer should replay with:

```
SEARCHING FOR CREATE
LOADING
READY
```

4.) Now, type the following:

```
RUN
```

followed by a carriage return.

5.) The following prompt appears:

```
INSERT FORTRAN-64 DISK
AND PRESS RETURN
```

6.) Now press the Return key All library and runtime routines are read from the disk. An asterisk appears to show you the progress.

7.) When complete, a new prompt appears as follows:

```
INSERT DATA DISK
AND PRESS RETURN
```

8.) Insert the new data disk and press the Return key. A new prompt appears as follows:

```
DO YOU WANT TO FORMAT? (Y/N CR-N)
```

If Y is entered this prompt appears:

```
ENTER DISK NAME:
```

Type a new disk name and press the Return key.

A second prompt then appears as follows:

```
ENTER DISK ID:
```

The new disk ID (not greater than 2 characters) can then be entered and a carriage return is depressed.

After formatting or if N or CR is depressed, all required libraries and runtime routines are copied to your disk.

9.) When complete a message is displayed as follows:

```
DATA DISK CREATED
```

10.) At this point, source programs can be stored on the data disk along with any user-defined libraries. You can now start Fortran-64.

Note: If any disk errors are detected during the create process, they are displayed on the screen as follows:

```
DISK ERROR xx FOUND
```

where xx is the error number (see your disk drive manual for error explanations).

STATEMENT DESCRIPTIONS

In the following pages are the statements recognized by Fortran-64. The descriptions consist of a short explanation of the statement, syntax for the statement, examples and notes for its use. The syntax shows, in a condensed form, the structure of the statement. To expand the syntax into a usable form the following rules must be followed:

1. Capitalized words and punctuation should be used exactly as shown.

2. Small lettered words represent place holders for user supplied syntax.

3. All spaces, except spaces between quotes, are optional. For example the following two statements are equivalent:

```
IF(I.EQ.J)GOTO 10
I F ( I . E Q . J ) GO TO 1 0
```

4. The following symbols have special meaning:

[ ]	syntax contained within is optional.
{ }	syntax contained within can be repeated.
|	OR - one or the other symbol on either side of the | can be used at this position.

Examples:

Syntax	Expansion:
`CLOSE filenumber{[,filenumber]}`	`CLOSE 8`
	`CLOSE 8,9,10`
`WRITE(filenumber,*|+)`	`WRITE(4,*)`
	`WRITE(6,+)`

5. Variable names are up to 5 characters long. They must begin with an alphabetical character (A-Z)

ARITHMETIC EXPRESSION

PURPOSE: To allow the calculation of arithmetic equations usin variables and constants.

SYNTAX:
```
varnam = varban|const|funcnan {[ope
varnam|const|funcnam]}
```

where: `varban` is a simple variable or array element.
`const` is a constant of the same type as th variables being used.
`funcnam` is the name of a function being invoked.
`oper` is a mathematical operator.

EXAMPLES:
```
X = 1.0
IY = ((J + 4) * 3) / 6 * 4
A = SIN(X) + 6.0
```

NOTES:
1) Mixed mode operations are not permitted (reals and integers mixed).
2) Variables must be explicitly converted using `IFIX` or `FLOAT`.
3) For a list of operators see Appendix A.
4) All variables on the right side of the equal sign must have been previously defined in the program.

CALL

PURPOSE: To transfer program control to the specified subroutine.

SYNTAX: `CALL subname {varnam|arithmetic expression|constant {[,varnam| arithmetic expression|constant]}}`

where `varnam` is a simple variable or array element.
`subname` is the subroutine name up to 5 characters long.

EXAMPLES:
```
CALL SUB1 (X,Y)
CALL TEXT (1,3,X,I)
CALL XXX (X,1,I+6)
```

NOTES:
1) Values in parameters can be passed to and from the subroutine.
2) Entire arrays cannot be passed as parameters. (See the COMMON statement)

CALL EXEC

PURPOSE: To allow direct access to Kernal or user written machine language subroutines.

SYNTAX: `CALL EXEC (ia,ix,iy,istat,iaddr)`

where: `ia, ix, iy, istat` are integer variables or constants with values 0 to 255.
`iddr` is an integer variable or constant with a value 0 to 65535.

EXAMPLES:
```
CALL EXEC (IA,IX,IY,ISTAT,LDTIM)
CALL EXEC (6,3,8,1,64233)
CALL EXEC (0,0IVAL1,IVAL2,IADDR)
```

NOTES:
1) `ia`, `ix`, and `iy` represent the contents of the internal 6510 registers A, X, and Y before the call.
`istat` represents the 6510 STATUS register. If variables are used instead of constants in these positions, they will contain the new values of the A,X,Y, and STATUS registers, after returning from the subroutine.

CLOSE

PURPOSE: To close a file previously opened for communication.

SYNTAX: `CLOSE filenumber ([,filenumber])`

where: `filenumber` is a constant or variable.

EXAMPLES:
```
CLOSE I,J
CLOSE 15
```

NOTES:
1) Performs the same function as in BASIC.
2) The filenumber must have been defined in an `OPEN` statement.

COMMENTS

PURPOSE: To allow embedded documentation in programs.

SYNTAX: `* any characters`

EXAMPLES:
```
* This is a comment line
* comments make programs more readable
```

COMMON

PURPOSE: To allow arrays in different subprogram units to share the same block of memory.

SYNTAX: `COMMON`

EXAMPLES:
```
COMMON
INTEGER IAXX (25), C (10)
REAL XXXX(4)
CHARACTER NAME (15)
ENDCOMMON
```

NOTES:

1) The `COMMON` statement signals the beginning of an array block that can be shared by many subprogram units. The `ENDCOMMON` statement ends the block of common.

2) The `COMMON` block can be used to equivalence two different data types between two program units. An example, an array of 5 integers in a subroutine can be treated as an array of 10 characters in another subroutine.

CONTINUE

PURPOSE: To allow block structuring of a program.

SYNTAX: `[label] CONTINUE`

where: `label` is an unsigned integer constant.

EXAMPLES:
```
100 CONTINUE
CONTINUE
DO 10 I = 1 , 10
      .
      .
10 CONTINUE
```

NOTES:
1) The CONTINUE statement can be used for clarity.
2) The CONTINUE statement should be used to end a DO block.

DATA

PURPOSE: To assign values to a variable.

SYNTAX:
```
DATA varnam/constant/|/'string'/
{[,varnam/constant/|/'string'/]}
```

where: varnam is a simple variable or array element or array name.
constant is a value to be assigned to the variable.

EXAMPLES:
```
DATA A/1.26/,I/29/
DATA X/'test'/,J/6*327/
DATA I(6)/1,2,4,8/
DATA Z/.TRUE./,Y/.FALSE./
```

NOTES:
1) There are no restrictions on where a DATA statement can appear in a program. In fact, to conserve memory, it is suggested that a DATA statement be used to replace a simple assignment statement.
2) The following multiple assignment is not permitted:
```
a,b,c/1.2,3.45,1.9
```
3) The * symbol can be used as a repeat specifier to fill an array more efficiently, as in the example above. As shown, the six elements of J would contain 327.

DIMENSION

PURPOSE: To define the dimensions and bounds of arrays.

SYNTAX:
```
DIMENSION varnam(bounds)
{[,varnam(bounds)]}
```

where: `varnam` is an array variable.
`bounds` is the integer dimension.

EXAMPLES:
```
DIMENSION J(2,6),A(6)
DIMENSION K(10)
```

NOTES:
1) 1 or 2 dimensional arrays are permitted.
2) An array can also be dimensioned using the type statement.

DO

PURPOSE: To allow a method of looping on a group of statements.

SYNTAX:
```
DO label varnam = is,if[,ic]
```

where: `label` is an unsigned integer.
`varnam` is a simple variable.
`is` is a signed integer constant or variable (starting value).
`if` is a signed integer constant or variable (final value).
`ic` is a signed integer constant (increment)

EXAMPLES:
```
DO 10 J=1,10
DO 200 K=I,J,2
DO 60 I=0,-5,-1
```

NOTES: 1) The integer specified as a label should correspond to a labeled continue statement used later in the program.
2) DO loops can be nested.

ELSE

PURPOSE: To allow one of two blocks of statements to be executed depending on a conditional statement.

SYNTAX:
```
IF (conditional) THEN
   statement
      .
      .
   statement
      .
      .
   statement
 ENDIF
```

EXAMPLES:
```
IF (TIME .EQ. 0.0) THEN
   WRITE (3,*) 'TIME IS UP.'
   TIME = 100.00
ELSE
   WRITE (3,*) 'WAITING...'
   TIME = TIME - 1.0
ENDIF
```

NOTES: 1) The ELSE block can contain IF statements to further the decision making process.
2) The ELSE must be used in conjunction with the IF THEN and ENDIF statements.

ENDIF

PURPOSE: To terminate the end of a block IF statement.

SYNTAX:
```
ENDIF
```

EXAMPLE:
```
IF (X.EQ. 1) THEN
   WRITE (3,*) X
ENDIF
IF (X.EQ. 6) THEN
   WRITE (3,*) X
ELSE
   WRITE (3,*) '*'
ENDIF
```

NOTES:
1) ENDIF is required at the end of any compound IF statement.
2) When nesting compound IF statements within blocks each must have a corresponding ENDIF.

END

PURPOSE: To indicate the end of a program unit.

SYNTAX:
```
END
```

EXAMPLES:
```
PROGRAM TEST
COMMON
INTEGER X(5),J(20)
ENDCOMMON
READ (0,*) I
CALL SUB1 (I)
WRITE(3,*)X(1)
STOP
END
SUBROUTINE SUB1 (J)
COMMON
```

```
INTEGER I(10),A(15)
ENDCOMMON
I(1) = J
RETURN
END
```

NOTES: 1) END is required at the end of all program units. (Program, Subroutine or Function)

ENDCOMMON

PURPOSE: To signal the end of a common block of array storage.

SYNTAX: ENDCOMMON

EXAMPLES:
```
COMMON
INTEGER XXX(10)
ENDCOMMON
```

NOTES: 1) See the COMMON statement for general comments.

FUNCTION

PURPOSE: To define the beginning of a function subprogram.

SYNTAX: FUNCTION name (varnam|[,varnam]))

where: varnam is a variable or array element.
name is the function name up to 5 characters.

EXAMPLES: FUNCTION SUB1 (X,Y)

NOTES: 1) Expressions are not allowed as arguments.
2) Entire arrays cannot be passed as parameters. (See the COMMON statement)

3) An argument is always required even if it's a dummy argument.
4) The FUNCTION is invoked by name in an expression as follows:

```
x 3 + sub1(i,j)
```

5) The function name must occur on the left side of an assignment statement, somewhere within the subprogram body.
6) The function only returns one value, all other arguments remain unchanged.
7) See Appendix B for a list of build-in functions.

FORMAT

PURPOSE: To structure input and output information

SYNTAX: `label FORMAT(desc{[,desc]})`

where: `desc` is a format descriptor.
`label` is an unsigned integer.

EXAMPLES:
```
100 FORMAT(I3,5F12.3)
200 FORMAT(1X,'ANSWER = ',3(2X,I4,L2))
```

NOTES:
1) See Appendix C for a list of format descriptors.
2) Parenthesis can be used to nest format descriptions as shown in the second example above.

computed GOTO

PURPOSE: To allow branching to a labeled statement depending on the result of an integer expression.

SYNTAX: `GOTO (label{[,label]}) I`

where: `label` is an unsigned integer.
`I` is an integer expression.

EXAMPLES:
```
GOTO (10,20,30) J
GOTO (10,20) (I+3)/6
```

NOTES:
1) In the first example, if `J=1` then branch is made to 10, of `J=2` then a branch is made to 20, etc.
2) If the expression is less than 1, a branch is made to the first label. If it is greater than or equal to the number of labels, a branch is made to the last label.

unconditional GOTO

PURPOSE: To allow branching to a labeled statement.

SYNTAX: `GOTO label`

where: `label` is an unsigned integer.

EXAMPLES:
```
GOTO 10
GOTO 999
```

NOTES:
1) The statement immediately after the `GOTO` statement must have a label to allow access to that statement.

arithmetic IF

PURPOSE: To allow control to pass to one of three different labeled statements depending on the evaluation of an arithmetic expression.

SYNTAX: `IF (express) label, label [,label]`

where: `express` is an arithmetic expression.
`label` is an unsigned integer.

EXAMPLES:
```
IF (I) 10,20,30
IF (X-3) 10,20
IF ((SIN(X) + 2) / 3) 30,25,100
```

NOTES:
1) Transfer to first label if expression is < 0.
2) Transfer to second label if expression is = 0.
3) Transfer to third label if expression is > 0.
4) When the last label is omitted, transfer goes to the second label when the expression is >= 0:

The following two examples are equivalent:

```
IF (I) 10,20
IF (I) 10,20,20
```

logical IF

PURPOSE: To allow action to be taken depending on a conditional expression within the `IF`.

SYNTAX: `IF (logical expression) statement`

where: `logical` expression - RE.LO.RE
RE = arithmetic expression
RO.= arithmetic expression
RO = relational operator
relational operators are:

.EQ. - equal
.NE. - not equal
.LT. - less than
.GT. - greater than
.GE. - greater than or equal

LO = logical operator
logical operators are:
.OR. - or
.AND. - and
.NOT. - not

EXAMPLES:
```
IF (X.EQ. 0) GOTO 10
IF (I .LT. J+2) WRITE (4,*) I
IF ((I .EQ. 1) .AND. (J .EQ. 2)) READ (0,10)
A,B
```

NOTES: 1) The following statements can only be used with `IF THEN` and cannot be part of logical `IF`;
a) `DO`
b) `CONTINUE`

IF THEN

PURPOSE: To allow conditional execution of a group of statements.

SYNTAX:
```
IF (logical expression) THEN
   statement
      .
      .
ENDIF
```

EXAMPLES:
```
IF (I .EQ. 0) THEN
   READ (0,*) NAME
   WRITE (3,*) 'name is', NAME
   I = I + 1
ENDIF
```

NOTES: 1) The ENDIF statement terminates the end of the block statements.

IMPLICIT

PURPOSE: To override or confirm the type associated with the first letter of a variable name.

SYNTAX:
```
IMPLICIT typnam (char|char_range)
{[,typnam(char|char_range)}]
```

where: typnam is any standard data type.
char is any letter of the alphabet.
char_range is a character range.

EXAMPLES:
```
IMPLICIT REAL (A,B), INTEGER (C-R),
LOGICAL (E)
```

NOTES: 1) The IMPLICIT statement can only be used once in any program module.
2) The type declaration can still override the type implicitly specified by the starting letter of a variable name.

MEM

PURPOSE: To examine or alter individual memory locations.

SYNTAX:
```
MEM (varnam|constant)-integer
expression
varnam = MEM (varnam|constant)
```

where: the argument between the () is an address between 1 and 65535
varnam is an integer variable.

EXAMPLES:
```
MEM (16763) = I + 6
I = MEM (0)
MEM (IADD) = 4
J = MEM (IADD) + 6 - 3 * (J+2) / MEM(J)
```

NOTES:
1) Using MEM on the left side of an equation is equiva lent to the poke in BASIC. Using MEM on the righ side of an equation is equivalent to the peek i BASIC.
2) Memory locations can only hold values between and 255. Only the low order byte of the integer i transferred to memory. Getting a value from memor stores a 0 in the upper byte of the integer variable.

OPEN

PURPOSE: To establish a connection between a file and a device

SYNTAX:
```
OPEN filenumber,devicenumber
[,secondary address
{,'filename'|name}]
```

where: `filenumber`, `devicenumber` and `secondary address` are constants or variables.
`'filename'` is the actual filename in quotes.
`name` is the filename stored in an array of type `CHARACTER`.

EXAMPLES:
```
OPEN I,J,K,L
OPEN I,J
OPEN I,J,K
OPEN I,J,K,' test file'
```

NOTES:
1) Performs the same function as BASIC.

2) Refer to a particular device manual for more information on the syntax of the OPEN statement for the device.

PAUSE

PURPOSE: To allow program execution to stop until a key on the keyboard is depressed.

SYNTAX: `PAUSE constant`

EXAMPLES: `PAUSE 148`

NOTES: 1) Refer to the Commodore keyboard codes to determine the value of the constant in the above syntax.

PROGRAM

PURPOSE: To name a program. After linking the program must be manually saved.

SYNTAX: `PROGRAM name`

EXAMPLE: `PROGRAM TEST`

READ. (free-format)

PURPOSE: To acquire data from the keyboard or from a file or device.

SYNTAX:
```
READ (filenum,*|#) ['string',]
varnam{[,varnam]}
```

where: `filnum` is an unsigned integer representing the file-number used in the OPEN statement.

varnam is a simple variable, array element or array name of type CHARACTER.

EXAMPLES:
```
READ (10,*) A,B,IX
READ (0,*0 'Enter a value: ',IY
READ (0,#) 'Enter Password',PASWD
```

NOTES:
1) A string to be used as a prompt can only appear as the first thing after the right parenthesis.
2) Filenumber 0 is reserved for the keyboard.
3) If a pound sign (#) is used in place of the asterisk (*), a no-echo read is performed.
4) Only arrays of type CHARACTER can be read in directly.
5) Logical values are read in as 'T' or 'F'.
6) A READ statement without variables skips a record.
7) A space is used as a delimiter between values of type INTEGER, REAL and LOGICAL.

READ (formatted)

PURPOSE: To acquire data from the keyboard or from a file or device according to a specified format.

SYNTAX:
```
READ (filenum,label) varnam{ [,varnam] }
```

where: filenum is an unsigned integer representing file-number used in the OPEN statement.
varnam is a simple variable, array element or array name of type CHARACTER.
label is an unsigned integer constant.

EXAMPLES:
```
READ (10,100) A,B,IX
100 FORMAT (2F8.2,I6)
```

NOTES:
1) Filenumber 0 is reserved for the keyboard.

RETURN

PURPOSE: To transfer control from a subprogram back to the calling program unit.

SYNTAX: `RETURN`

EXAMPLES:
```
RETURN
IF (x .LT. 0.0) RETURN
```

NOTES: 1) Subprogram units must have at least one `RETURN` statement.

STOP

PURPOSE: To allow program flow to cease and execution return to BASIC.

SYNTAX: `STOP`

EXAMPLES:
```
STOP
IF (ix .LT. 0) STOP
```

NOTES: 1) `STOP` should only be executed within a main program unit.

SUBROUTINE

PURPOSE: To define the beginning of subroutine subprogram.

SYNTAX: `SUBROUTINE subnam (varnam{[,varnam]})`

where: `varnam` is a variable.
`subnam` is the subroutine name up to 5 characters.

EXAMPLES: `SUBROUTINE SUB1 (x)`

NOTES:
1) The argument list determines two things about passing values:
type of argument - determined by 1st character of `varnam`
placement of argument - determines placement of parameters in a `CALL` statement.
2) An argument is always required even if it's a dummy argument.
3) A subroutine is invoked by the `CALL` statement.

TYPE STATEMENTS

PURPOSE: To specify the type of variables listed in the statement.

SYNTAX: `type varnam{[,varnam]}`

where: `type` is the variable type such as `INTEGER`, `REAL`, `CHARACTER` or `LOGICAL`.
`varnam` is a variable definition.

EXAMPLES:
```
INTEGER STATE, COUNT, ZIP
REAL COST, VALUE
LOGICAL FLAG
CHARACTER CH
INTEGER K(10)
```

NOTES:
1) This declaration overrides the type implicitly specified by the `IMPLICIT` statement.
2) An array can be dimensioned with a type statement.
3) This declaration assigns an explicit type to symbolic names that would otherwise have their type implicitly determined by the first letter of their names.
4) See Appendix C for data types.

WAIT

PURPOSE: To allow program execution to be suspended for a specified time period.

SYNTAX: `WAIT n`

where: n is an integer constant.

EXAMPLE: `WAIT 2`

NOTES: 1) Resolution is in seconds.

WRITE (free-format)

PURPOSE: To transfer data from memory to the screen, printer or file.

SYNTAX: `WRITE(filenum,*|+){['string'|varnam|/decimal number,}]`

where: `filenum` is an unsigned integer representing the filenumber used in the `OPEN` statement.
`varnam` is a simple variable or array element or array name of type `CHARACTER`.
`/decimal number` sends the single byte equivalent of the decimal number.

EXAMPLE:
```
WRITE (10,*) 'The answers are: ',A,B,IX
WRITE (3,*) /147,IX,' is the value.'
WRITE(3,+) 'ENTER YOUR PASSWORD:'
```

NOTES:
1) Filenumber 3 is reserved for the screen and need not be opened previously.
2) Only arrays of type `CHARACTER` can be written out directly.
3) Logical values are written out as T or F.

4) A WRITE statement without variables isn't permitted.
5) The following field lengths are used on outputs:

REAL	12 Columns
INTEGER	6 Columns
LOGICAL	1 Column
CHARACTER	1 Column
CHARACTER array	number = size of array

6) If the '+' symbol is used instead of the '*' symbol, a no return WRITE is performed.

WRITE (formatted)

PURPOSE: To transfer data from memory to the screen, printer or file according to a specified format.

SYNTAX:
```
WRITE (filenum,label) variable
{[,variable]}
```

where: filenum is an unsigned integer representing the filenumber used in the OPEN statement.
variable is a simple variable or array element or array table.
label is an unsigned integer constant.

EXAMPLES:
```
WRITE (10,100) A,B,IX
100 FORMAT(2F8.2,16)
```

NOTES:
1) Filenumber 3 is reserved for the screen and need not be OPENed first.
2) Filenumber 4 and 5 are reserved for the printer.
3) Filenumber 8,9,10,11 and 15 is reserved for the disk drive.
4) Only arrays of type CHARACTER can be written directly.

5) A `WRITE` statement without a label referencing a format skips a record or line.

OPERATORS

+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Integer Exponentiation (see built in functions for real exponentiation)

APPENDIX A

BUILT-IN FUNCTIONS

Trigonometric functions: (trig.lib)

SIN (x)	Sine of an angle given in radians
COS (x)	Cosine of an angle given in radians
TAN (x)	Tangent of an angle given in radians

Arithmetic functions: (func.lib)

EXP (x)	Calculates e to the x power
ALN (x)	Calculates natural log of x
LOG (x)	Calculates log to base 10 of x
ABS (x)	Calculates REAL absolute value of x
IABS (x)	Calculates INTEGER absolute value of x
POW (x,y)	Calculates x to the y power, where x and y are REAL
FLOAT (i)	INTEGER to REAL conversion
IFIX (x)	REAL to INTEGER conversion
MOD (i,j)	MODULO; Calculates INTEGER remainder
AMOD (x,y)	Calculates REAL remainder

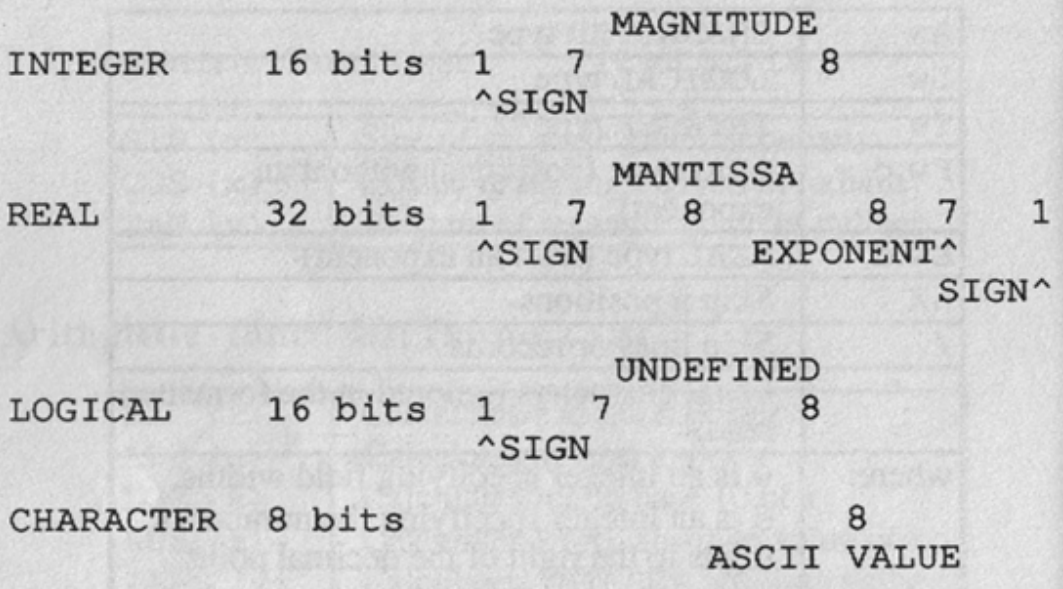
APPENDIX B

FORMAT DESCRIPTOR

Aw	CHARACTER type
Lw	LOGICAL type
Iw	INTEGER type
Fw.d	REAL type (positional without an exponent)
Ew.d	REAL type (with an exponent)
nX	Skip n positions
/	Skip lines or records
' '	literal characters (ignored in the formatted read)
where:	w is an integer specifying field widths. d is an integer specifying the number of digits to the right of the decimal point. n is an integer.

APPENDIX C

DATA TYPES



APPENDIX D

ERROR DESCRIPTIONS

error message format: `*****ERROR xxx yyy*****`

where: `xxx` = error number from list below
`yyy` = character position in line that caused the error (not including spaces)

Examples:
```
IF (x .LT. 1.0 GOTO 10
*****ERROR 008 012*****
```

Error number	Description
0	illegal alphanumeric
1	slash missing
2	constant missing
3	illegal type
4	quantity of elements exceeds size of array
5	array too large
6	array not dimensioned/variable has no initial value
7	element subscript outside boundary of dimensioned array
8	parenthesis missing
9	bad repeat specifier
10	invalid real number
11	sign missing
12	exponent too large
13	exponent missing
14	number or label too large
15	comma missing
16	value not assigned to identifier
17	filename too large

Error Number	Description
18	invalid real descriptor in format
19	only 2 dimension array permitted
20	first statement or END missing
21	quote missing
22	symbol table overflow
23	invalid identifier name
24	identifier already defined
25	implicit statement used more than once
26	format not found
27	format too long
28	data memory requirements too large
29	object file too big
30	no corresponding label for DO loop
31-49	undefined
50	label not found
51	constant must be integer
52	'=' expected
53	identifier not found
54	increment cannot be zero
55	DO's nested too deep
56	type mismatch
57	invalid operator
58	parameter expected
59	'if' nested too deep
60	<cr> <Return> expected
61	corresponding 'IF not expected
62	SUBROUTINE expected
63	comma expected
64	integer variable expected

INDEX

GEOS INFO

Another Abacus Best Seller!

GEOS Inside and Out

If you use GEOS then our new book, ***GEOS Inside and Out***, has the info you need.

A detailed introduction is laid out for the novice–beginning with how to load the GEOS operating system...how to create a backup...how to alter the preference manager...how to format disks...learn geoWrite and geoPaint in detail...use geoPaint for designing floor plans or drawing electronic diagrams. Easy-to-understand examples, diagrams and glossary are included to enlighten the beginner.

The advanced user will find more detailed information on GEOS's internals and useful tricks and tips. Add a constant display clock–includes assembly and BASIC listing...complete listing of our **FileMaster** utility (converts your programs to GEOS format with an icon editor) with a line by line explanation...create a single-step simulator for observing memory and the various system registers...learn about windows and how to use them to your advantage...understand GEOS file structure.

If you're just getting started with GEOS or getting to the point of wanting to add your own applications, then ***GEOS Inside and Out*** will help you on your way. $19.95

Coming Soon!

GEOS Tricks & Tips

Continuing the tradition established by our famous C-64 reference library, ***GEOS Tricks & Tips*** is a collection of helpful techniques for anyone who uses GEOS with their Commodore. It's easy to understand without talking down to the reader, and detailed in the applications of the routines. Includes a font editor to create up to 64 point text and a machine language monitor. A perfect companion volume to ***GEOS Inside and Out***. Available Second Quarter. $19.95

GEOS, geoWrite, geoPaint are tradenames of Berkeley Softworks.

To receive your copy of ***GEOS Inside and Out*** and/or ***GEOS Tricks & Tips***, call now for the name of the dealer or bookstore near you. Or order directly using your Visa, MC or Amex card. Add $4.00 per order for shipping and handling. Foreign orders add $10.00 per book. Call or write today for your free catalog. Dealer inquires welcome—2000 nationwide.

Order both today!

You Can Count On **Abacus**

5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Just a few of our books...

Anatomy of the C-64
Insider's guide to 64 internals. Graphics, sound, I/O, kernal, memory maps, and much more. Complete commented ROM listings. 300pp $19.95

Anatomy of the 1541 Drive
Best handbook on this drive, explains all. Filled with many examples programs, utilities. Fully commented 1541 ROM listings. 600pp $19.95

Tricks & Tips for the C-64
Collection of easy-to-use techniques: advanced graphics, improved data input, DP/M, enhanced BASIC, disk handling and more. 275pp $19.95

Peeks & Pokes for the 64
Includes in-depth explanations of PEEK, POKE, USR, and other BASIC commands. Learn the "inside" tricks about your 64. 200pp $14.95

Graphics Book for the C-64
Best reference, covers basic and advanced graphics. Sprites, Hires, Multicolor, 3D-graphics, IRQ, CAD, projections, curves. 340pp $19.95

C-128 INTERNALS
Important C-128 information. Covers graphic chips, MMU, I/O, 80 column graphics and fully commented ROM listings, more. 500pp $19.95

1571 INTERNALS
Essential reference. Internal drive functions. Explains various disk and file formats. Fully-commented ROM listings. 450pp $19.95

C-128 TRICKS & TIPS
Fascinating and practical info on the C-128. 80-col hires graphics, bank switching. 300 pages of useful information for everyone. $19.95

C-128 PEEKS & POKES
Dozens of programming quick-hitters. Techniques on the operating system, stacks, zero page, pointers, and BASIC. 240pp $16.95

C-128 BASIC 7.0 Internals
Get all the inside info on BASIC 7.0. This exhaustive handbook is complete with fully commented BASIC 7.0 ROM listings. $24.95

Call now for the name of your nearest dealer. Or order direct with your credit card by calling 616/241-5510. Add $4.00 per order for S&H. Foreign add $12.00 per book.

Other books and software also available. Call or write for your free catalog. Dealers inquires welcome—over 2000 nationwide.

Commodore 64 and Commodore 128 are trademarks of Commodore Int.

Abacus

Abacus Software
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Selected Abacus Products for Commodore computers

BASIC

Complete BASIC compilers and development systems for the C-64 or C-128

"The package is easy to use and the manual well-written. It should take only a few minutes to create code from scratch, assuming the BASIC source code already exists... In summary, BASIC enhances the performance of programs written in BASIC. It provides a good introduction to those programmers who intend to go on to use larger machines and other high-level langiuages. I enjoyed using it."

—Shlomo Ginsberg
Commodore Microcomputers

BASIC 64 and **BASIC 128** are complete development systems that compile standard Commodore BASIC programs into either superfast machine code or very compact speedcode. In fact, the user can mix the two in during a single compilation. **BASIC-64** and **BASIC-128** speed up BASIC programs from 5 to 35 times faster.

BASIC lets the user compile a series of programs using the overlay features, and even allows the use of many of the language extensions found in Simon's Basic, Video Basic, Victree or BASIC 4.0.

BASIC-64 and **BASIC-128** compile to either ultra-fast 8510 machine code, very compact p-code, or a combination of both. There are two separate optimization levels. The user chooses the level suited to his specific needs. **BASIC-128** has faster and higher-precision math functions. It uses integer and formula optimizing techniques and is completely compatible with Commodore BASIC 2.0/7.0.

The 80-page programmer's guide explains the compiler's simple operation. For more in-depth use, it also covers the extensive compiler options and directives, flexible memory usage, program overlay techniques, optimization considerations and programming tips and hints so the user can understand every feature of this quality product.

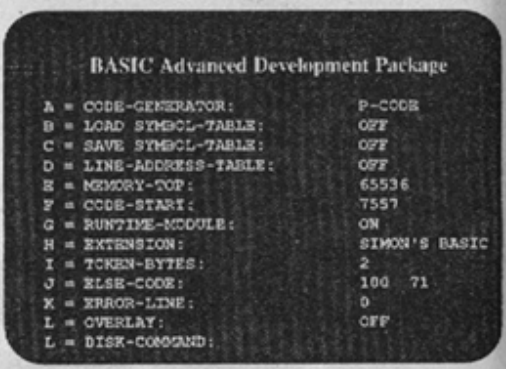
BASIC-128 was crafted in West Germany by one of the most successful author and compiler writers in Europe, Thomas Helbig. Our BASIC-64 and BASIC-128 packages are the tools users need to make their BASIC programs run lightning fast, and protect their programs from unwanted listing or alteration.

Make your BASIC programs

ZOOM

Convert them to high-speed machine language

BASIC Compiler



Hardware requirements:

BASIC-64:
Commodore 64 with 1541 or 1571 disk drive

BASIC-128:
Commodore 128 with 1541 or 1571 disk drive (supports 40- or 80-column monitor)

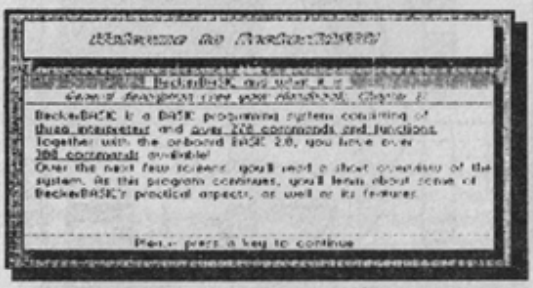
Printer optional

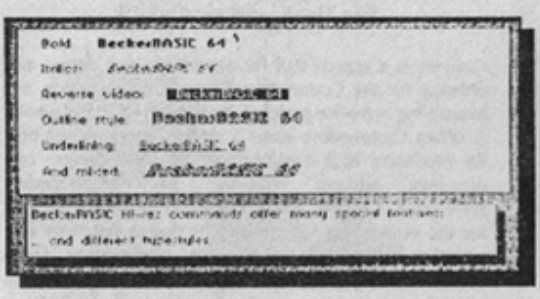
Suggested retail price:	
C-64 version	$39.95
C-128 version	$59.95

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

BeckerBASIC for GEOS

Now you can write BASIC applications to work with GEOS





Introducing **BeckerBASIC**. If you already know BASIC, you can now write your own GEOS applications in BASIC, easily.

BeckerBASIC gives you the power of over 270 new commands and functions.

Over 20 commands to make your programming easier. For example, TRACE, RENUMBER, DUMP, DIR, etc.

Packed with over 50 commands for easy disk access. Load and save blocks of memory or selected lines of your program. You can even PEEK and POKE into your disk drive's memory.

10 commands can be used for easier cursor control. Turn the cursor on and off. Set how quickly it flashes. Position it at any location on the screen.

20 commands are available for all your hires programming needs. Creat boxes, plot points, and draw lines.

18 additional commands are dedicated to creating sound. Set ring modulation, change the filter, alter the waveform and set the envelope.

Over 35 commands let you create and animate sprites with ease. Load and save sprites directly. Alter their size, change their positions and check for collisions. Use the sprite editor to create sprites and icons.

Use the *Pulldown Menu Construction Set* and *Dialog Box Construction Set* to aid in the creation of you own applications

Royalty-free distribution of your **BeckerBASIC** applications.

Now anyone can create applications in BASIC to run with GEOS. Only $49.95

For credit card orders call 1-616-698-0330

Call today or mail the coupon for your free catalog covering our complete line of software and books for the Commodore 64 and 128. Or ask for the location of the dealer nearest you. You can order direct by phone using your VISA, American Express or MasterCard or detatch and mail your completed coupon. Dealer inquiries welcome—over 2400 nationwide.

Abacus

5370 52nd Street SE
Grand rapids, MI 49508
Telex 709-101 • FAX 616/698-0325

If your Commodore dealer doesn't carry Abacus products, then have him order them for you. Or you can order direct using the following order blank or by calling—(616) 698-0330

Qty.	Product	Price	Total
_____	BeckerBASIC for the Commodore 64	$49.95	_____

In USA add $4.00 for S & H per order. Foreign add $12.00 per item. _____
Michigan residents include 4% sales tax _____
Total amount enclosed (US funds) _____

Payment: ( ) MasterCard ( ) VISA ( ) American Express
( ) Money Order ( ) Check

Card No._____ Exp._____
Name_____
Address_____
City_____ State_____ Zip_____
Phone No._____

GEOS is a trademark of Berkeley Softworks.
Commodore is a trademark of Commodore Electronics Ltd.

Selected Abacus Products for Commodore computers

Cadpak

Enhanced and 1351 Mouse Versions!

Computer-Aided Design package for the C-64 or C-128

Cadpak is a superb tool for computer aided design and drawing for the Commodore 64 and 128—it's been our bestselling software package for the last year and a half. It offers Commodore users a simple, versatile solution for producing high quality computer-aided designs and drawings without requiring any programming knowledge. **Cadpak** was designed to be simple enough for the novice, yet incorporate the design functions and printout capabilities of a truly professional CAD system. Its simplicity, accuracy and speed make **Cadpak** a standout.

Cadpak can be used with either the keyboard, an optional lightpen or optional mouse to draw directly on the screen and create and edit pictures, drawings, layouts and renderings. The feature that sets **Cadpak** apart from its competition is its exclusive **dimensioning** feature, which allows exact scaled output of designs to most popular dot-matrix printers (listed below). Choose from the menu options and draw on the screen at an exact location with **Cadpak's** exclusive *AccuPoint* cursor positioning.

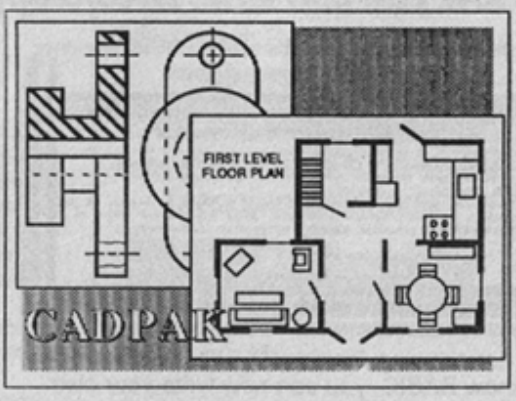
Cadpak's menu options make it easy to use for beginners. **Cadpak** also boasts many sophisticated features for the advanced user. Using the two graphics screens, you can draw lines, boxes, circles, ellipses; fill with solid colors or patterns; draw freehand; copy sections of the screen. The user can zoom in to do detailed design on a small section of the screen. **Cadpak's** improved object editor lets the user define and save furniture, electronic circuitry, machinery, etc. as intricate as the screen resolution permits. Perfect for all design needs on the Commodore C-64, 64C and C-128.

Cadpak-64 has two screens with 320 x 200 resolution. **Cadpak-128** has a first screen resolution of 640 x 360 and the second screen resolution of 320 x 200.

Hardware requirements: (Lightpen and mouse optional)
Cadpak 64:
Commodore 64
1541 disk drive (or MSD disk drive).

Cadpak 128:
Commodore 128
1571/1541 disk drive (or MSD disk drive)

1351 Mouse version now available!



Exclusive **Dimensioning** feature assures exact scaled output of designs

Cadpak Features:

- Precision scaled output to most dot-matrix printers —objects retain exact proportions when printed
- Two screens for flexible copy operations
- Drawing using either the keyboard, high-quality lightpen (optional), or new MOUSE version (optional, available April 87)
- Pre-defined or user-defined fill patterns
- SAVE/RECALL designs to and from diskette
- Library contains pre-defined objects, symbols, fonts—create and add custom symbols/fonts
- Mirror, rotate and zoom functions
- Advanced labeling/ measuring features
- Sophisticated graphic functions for lines, boxes, ellipses, arcs, etc.
- Custom-created text fonts or graphic symbols

Printers:

- Commodore 1525, 1526
- Comrex CR-220
- Epson MX, FX, Homewriter 10 and compatibles (Star Gemini SG-10, 10x, 10C, 15x, Panasonic KXP 1080)
- MPS 801, 802, 803, 1000
- Okidata Microline
- Okimate-10 b/w and color.
- Prowriter 8510A, 8510SC color
- Seikoska 1000
- Siemens PT88/89

Suggested retail price:

C-64 version	$39.95
C-128 version (Mouse version July 87)	$59.95

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

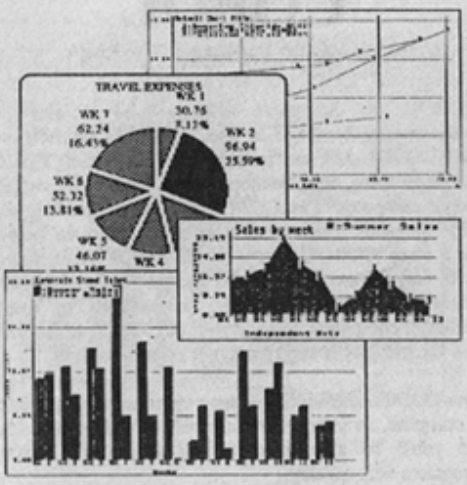
Selected Abacus Products for Commodore computers

Chartpak

Professional charting & graphing package for the C-64 or C-128

Charts and graphs are the most effective method of representing statistical data from business and science in a comprehensible, easy to digest format that quickly and accurately conveys numerical information.

Chartpak 64 or **128** makes it simpler than ever before for the professional who wants to easily create high-quality charts and graphs without any time-consuming programming. **Chartpak's** defaults let the user build professional quality charts or graphs right off the bat, or charts and graphs can be built to specification by selecting from **Chartpak's** easy-to-use menus. Every option is menu-driven—all the user has to do is enter the required data, choose the chart format and then watch the chart as it's drawn.



Chartpak quickly draws any one of 8 different formats of pie, bar, line and scatter graphs. Since **Chartpak** is an interactive software tool, changing a feature in the **Chartpak** graph is no problem. The user can immediately change the scaling, labels, axis, colors or bar-filling options at any time. **Chartpak** also has built-in features for statistical functions: least squares, regression, mean and exponential smoothing, and can be added to chart and graphs.

The 140-page manual contains several tutorials to walk the user through the easy process of building charts and graphs using extensive examples and sample charts. In addition, the screen menus are identified and cross-referenced to the user's guide for added convenience and ease of use.

When the user's created a chart or graph to satisfaction, they can get a hardcopy of it with most popular dot-matrix printers in either of two sizes. Many **Chartpak** users have reproduced these charts and graphs for reports and presentations.

The C-128 version that takes advantage of the added features and extra memory of the Commodore 128 computer. More data can be entered to build charts, and charts and graphs can contain more detail. In addition, **Chartpak-128** gives 3 times the resolution of the 64 version. This permits an entire chart or graph to be previewed on the screen, or it can be scrolled to show the higher resolution detail.

Chartpak 64 and Chartpak 128 Features:

- Enter data manually, or use data straight from Busicalc 1, Multiplan and Calc Result
- Draws pie charts, vertical or horizontal line charts, vertical or horizontal grouped, mixed or stack bar charts, scatter diagrams
- Statistical routines include average, standard deviation, least squares, two-dimensional data, exponential smoothing
- Easy to use menu operations
- Handles up to four data sets totaling 200 points (more with C-128 version)
- Saves data and/or chart specifications separately
- Complete manual with **Chartpak** data reduction tutorials
- Printout in two different sizes
- **Chartpak 128** has 3X the resolution of the '64 version

Hardware requirements:

Chartpak 64:
Commodore 64
1541 disk drive (or MSD disk drive).

Chartpak 128:
Commodore 128 with 40- or 80-column monitor
1571/1541 disk drive (or MSD disk drive)

Printers:
Commodore 1525 and 1526, MPS 801, Epson, Star Gemini, Okidata, Okimate, Siemens, others.

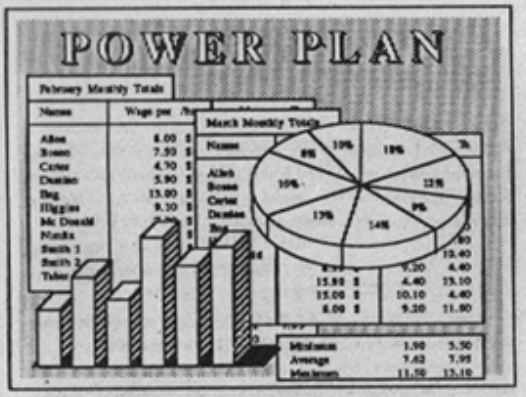
Suggested retail price:	
C-64 version	$39.95
C-128 version	$39.95

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Selected Abacus Products for Commodore computers

PowerPlan-64

Spreadsheet and Graphics package for the C-64



"PowerPlan is one of the best programs ever written for the Commodore 64, giving Lotus 1-2-3® a run for the money... It was a pleasure to work with this amazing product. I strongly recommend PowerPlan to anyone interseted in using spreadsheet programs for business."

—Al Willen
Commodore Magazine

Ever since VisiCalc and Lotus 1-2-3 stormed the personal computer market, the computer has become an important financial planning tool. By using an electronic ledger, you can perform hundreds of calculations and "what-if" analyses quickly and easily, and reduce reams of data into meaningful information.

PowerPlan-64 offers the '64 user a software tool that combines spreadsheet operations with a powerful built-in graphics program to display data in graphs as well as numbers. **PowerPlan-64** can handle up to 255 rows by 63 columns—a total of 16,065 individually protected cells. This outstanding package includes all major math functions, manually controllable calculation mode, and built-in disk commands. The integrated graphics program, PowerGraph, has eight different windows —you can select bar charts, curve graphs, point charts, pie charts, or min-max charts in two or three directions.

PowerPlan-64's menus make it easy to use for the first-time spreadsheet user. All of **PowerPlan-64**'s selections are clearly displayed on the screen for the user to choose from. In addition, online HELP screens are available at the touch of a key.

PowerPlan-64's complete 200-page handbook has a plain-speaking tutorial that gently introduces the user to spreadsheets.

Hardware requirements:

Commodore 64
1541 disk drive (or MSD disk drive)

Printer optional

Self-running demo available

PowerPlan-64 Features:

- Menus make PowerPlan-64 easy to learn
- Large capacity spreadsheet serves all the user's analysis needs
- Convenient built-in notepad documents user's important memos
- Flexible online calculator gives access to quick computations
- Powerful options such as cut, copy and paste operations speeds the user's work
- Integrated graphics summarize hundreds of data items
- Draws pie, bar, 3D bar, line and area charts automatically (8 chart types)
- Multiple windows emphasize the analyses

Printers:
PowerPlan-64 works with the following printers:

- Commodore 1525, 1526
- Epson MX, FX, Homewriter 10 and compatibles (Star Gemini SG-10, 10x, 10C, 15x, Panasonic KXP 1080)
- MPS 801, 802

Suggested retail price:
C-64 version **$39.95**

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330