TYMSHARE MANUALS

REFERENCE SERIES

ADDENDUM TO SUPER FORTRAN

JANUARY 1972

TYMSHARE, INC.
10340 BUBB ROAD
CUPERTINO, CALIFORNIA 95014

CONTENTS

INTRODUCTION

The Tymshare SUPER FORTRAN language is constantly being enhanced and improved to bring to the time-sharing FORTRAN user the best conversational FORTRAN language available. This addendum documents all new features added to the SUPER FORTRAN language since the publication of the *Tymshare SUPER FORTRAN Reference Manual* (April 1970) and supersedes all SUPER FORTRAN language bulletins published since that manual.

In addition, the SUPER FORTRAN compatibility mode of the new BATCH FORTRAN IV language is described. This feature permits programs written and debugged in SUPER FORTRAN to be executed in the faster BATCH FORTRAN IV operating system.

Often the SUPER FORTRAN programmer needs to debug a link in his program. A programming hint to facilitate such debugging is illustrated in this addendum.

In all examples in this document, everything typed by the user is underlined. Lower case letters used in an example of a command form represent the letters that are typed. For example, the characters *file name* in a sample command form indicate that a file name should be typed at that point.

The symbols for user-typed Carriage Returns, Line Feeds, Alt Mode/Escapes, and Emergency Exit Keys are:

Carriage Return:	↵
Line Feed:	↴
Alt Mode/Escape:	⊕
Emergency Exit Key:[1]	☆

Control characters are denoted in this document by a superscript c. For example, D^c denotes Control D.

1 - The keyboard position for the Emergency Exit Key varies among terminals. It is usually a Control Back Arrow or a Control Underscore. The character has ASCII code 159 (237 octal) and the internal code 127 (177 octal).

NEW SUPER FORTRAN ADDITIONS

FUNCTIONS

SUPER FORTRAN includes nine new functions: TEL, WAIT, ONLC, YEAR, EXEC, TCP, ESC, SETSIZE, and GOFILE; each is described below.

The TEL Function

TEL is a logical function with one argument. The argument is a dummy argument and may be of any type; it has no relation to the function value returned. TEL returns the value 1 (true) if there is terminal input waiting to be processed; otherwise, it returns the value 0 (false).

For example, suppose a program contains a DO loop that the user knows may take a long time to terminate. If this should happen, he wants to be able to print the value of the indexing variable at any time during the execution of the loop to check its progress. The following partial program demonstrates this.

```
 10            STRING A(3)
  .
  .
  .
 40            DISPLAY "NOW WE CALCULATE X"
 50            DO 100 I=1,K
  .
  .
  .
130            IF ((Y-X) .LT. 1E-8) GO TO 200
140            IF (TEL(0)) READ (0,3000) A .ELSE. GO TO 100
150            WRITE (1,1000) I
160      100   CONTINUE
170      200   WRITE (1,2000) X
180      1000  FORMAT (" I=",I4)
190      2000  FORMAT ("X IS ",F12.5)
200      3000  FORMAT (&,S1)
210            END
```

Any valid character typed by the user during the execution of the loop causes the current value of I to be printed.[1] The character typed is stored in A, not because it is going to be used, but to clear the input buffer and to prevent its being confused with later input. Then, since no further input is waiting, TEL is again false until another character is typed.

1 - The following characters are not acceptable with the & FORMAT: Control A, Control Q, Control W, Control D, Control V, Control J (Line Feed), and Control M (Carriage Return).

The use of this program and the resulting printout are shown below.

```
>RUN
NOW WE CALCULATE X
^CI=2425                    The user checked the value of I twice.
? I=4662                    The first time he typed a Control B; the
X IS     463.39056          second time he typed a question mark.

( 0210 )>
```

Adding the statement

151 PAUSE

would enable the user to continue in the loop by typing CONTINUE after the value of I is printed, or to change or print the values of some variables and then type CONTINUE.

The WAIT Function

The WAIT function causes SUPER FORTRAN to wait a specified number of seconds before proceeding with the program. The WAIT function must appear in an assignment statement. Its form is

variable = WAIT(n)

where the number of seconds, n, may be any numeric variable or expression.

If n is non-negative when the WAIT function is executed, SUPER FORTRAN pauses until the output buffer is cleared before waiting n seconds. Thus, if the user wishes to print an explanatory message or question before pausing n seconds, the message is completely printed before the wait begins. If n is negative, the wait of length equal to the absolute value of n begins as soon as the WAIT function is executed, whether or not the output buffer is empty.

Example

```
>LIST
  1             STRING A(3)
  2             WRITE (1,500)
  3      100    X=WAIT(10)
  4             IF (TEL(0)) GO TO 200 .ELSE. WRITE (1,600)
  5             ACCEPT A
  6             IF (A .EQ. "YES") WRITE (1,700) .ELSE. GO TO 300
  7             STOP
  8      200    ACCEPT B
  9             IF (B .EQ. 1513) GO TO 400
 10      300    DISPLAY "TRY AGAIN..."
 11             GO TO 100
 12      400    DISPLAY "VERY GOOD!!!"
 13      500    FORMAT ("IN WHAT YEAR DID BALBOA DISCOVER THE PACIFIC?",&)
 14      600    FORMAT (//"DO YOU GIVE UP? ",&)
 15      700    FORMAT ("THE YEAR WAS 1513.")
 16             END
```

```
>RUN↵
IN WHAT YEAR DID BALBOA DISCOVER THE PACIFIC?
                    Ten seconds pass.
DO YOU GIVE UP? NO↵
TRY AGAIN...
1541↵
TRY AGAIN...
1513↵
VERY GOOD!!!

(016 )>
```

The ONLC Function

ONLC is a utility function which enables and disables lower case mode. ONLC must appear in an assignment statement; its form is

variable = ONLC(n)

where the variable may be of any type. If the argument n is non-zero, lower case characters are accepted and printed by the program. If the argument is zero, lower case is disabled, and SUPER FORTRAN interprets all lower case alphabetic characters as upper case. For example,

```
Z = ONLC(1)        Lower case is activated.
:
Z = ONLC(0)        Lower case is disabled.
```

ONLC causes lower case characters to print as the corresponding upper case characters on a terminal without lower case. For example, the character 102 octal (lower case b) prints as a Line Feed without a Carriage Return on a Model 33 Teletype when ONLC is false. However, when ONLC is true, 102 octal prints as an upper case B on a Model 33 Teletype.

The YEAR Function

The YEAR function returns as its value the integer representation of the current year. The argument may be any numerical expression. Thus, Y=YEAR(0) returns in Y the value 1972. For example,

```
>10 Y=YEAR(0)↵
>20 DISPLAY Y↵
>30 END↵
>RUN↵
 1972
```

The EXEC Function

The EXEC function allows the user to execute any EXECUTIVE command from his SUPER FORTRAN program.[1] EXEC must appear in an assignment statement; its form is

variable = EXEC('command',r)

where command is any EXECUTIVE command enclosed in single quote marks, for example, 'WHY', 'DIR', and 'COPY'; a minimum of three characters must be specified. The second argument, r, can be any numeric expression; the variable on the left-hand side of the assignment may be of any type.

If r is non-zero, control returns to the next statement in the SUPER FORTRAN program after the EXECUTIVE command has been executed. If r is zero, control remains in the EXECUTIVE. In other words, SUPER FORTRAN considers a zero r to be equivalent to the QUIT command.

If the EXECUTIVE command specified requires further input (as in the COPY command), that input is accepted from the command file or with prompting from the terminal.

NOTE: The EXEC function cannot be used within a proprietary or remote file.

In the following example, the user gives the EXECUTIVE commands FILES, LIST, and MAIL. In each case, control returns to SUPER FORTRAN. Note that the MAIL command is abbreviated to the first three letters.

```
>LIST↵
 10            A=EXEC('FILES',2)
 15            WRITE (1,100)
 20            B=EXEC('LIST',2)
 25            WRITE (1,100)
 30            C=EXEC('MAI',-8)
 40      100   FORMAT(/)
 50            END
>RUN↵

SYM   ACCTS        The FILES command is executed.
SYM   ABC
SYM   DATA
SYM   INV
SYM   INV'STR.E'
SYM   PROG1

PROG1 INV'STR.E' INV DATA ABC ACCTS        The LIST command is executed.

NO MAIL.        The MAIL command is executed.

(@50 )>QUIT↵

-
```

1 - See the *Tymshare EXECUTIVE Reference Manual* for a description of the EXECUTIVE commands.

The TCP Function

TCP is a logical function which tests whether a command file is open. The form of the function is:

TCP(argument)

The argument may be any numerical expression; it is a dummy argument. TCP returns the value zero (false) if no command file is open and non-zero (true) otherwise.

The TCP function is useful in an IF statement to test whether a command file is open. For example,

IF (TCP(4).EQ.0) ACCEPT A .ELSE. GO TO 815

transfers control to statement 815 if a command file is open.

The ESC Function

The ESC function is used to disable and enable interrupts. ESC must appear in an assignment statement; its form is

variable = ESC(n)

where the variable may be of any type, and n is any numeric expression.

When ESC is executed with n equal to zero, interrupts are disabled. Thus, if the user types an Alt Mode/Escape or the Emergency Exit Key,[1] no action is taken. The system, however, records the fact that an interrupt was attempted and which kind it was. Multiple interrupts of either kind are recorded as only one interrupt; thus, if the user types two Alt Mode/Escapes and three Emergency Exit Keys after interrupts are disabled, only one of each is recorded.

If the program later executes an ESC function with non-zero n, interrupts are enabled and the following occurs:

1. If an Emergency Exit Key interrupt was recorded, control returns to the EXECUTIVE immediately.
2. If an Alt Mode/Escape was recorded and no Emergency Exit Key was recorded, the system checks to see if an ON INTERRUPT statement has been executed. If so, control transfers to the statement specified in the ON INTERRUPT statement. If no ON INTERRUPT statements have been executed, the recorded Alt Mode/Escape now interrupts execution in the normal manner.
3. If no interrupts were recorded, execution proceeds in the normal manner.

Whenever an ESC function is executed, the variable on the left-hand side of the assignment is set to zero (false) if no interrupt is waiting and to a non-zero value (true) if an interrupt is waiting. However, if there is an interrupt waiting when the interrupts are enabled with ESC and a non-zero argument, that interrupt is executed before ESC can return a value. Thus, in this one instance, the variable value remains unchanged from any previous assignment.

1 - The Emergency Exit Key is described on page 1.

Example

The following SUPER FORTRAN program is run three times. The first time no interrupts were typed. The second time the user typed an Alt Mode/Escape during the five-second wait in line 30, and the third time he typed the Emergency Exit Key during the five-second wait.

```
>LIST
 10            W=15
 15            ON INTERRUPT GO TO 100
 20            W=ESC(0)
 25            DISPLAY W
 30            D=WAIT(5)
 35            W=ESC(1)
 40            DISPLAY W
 45            GO TO 150
 50      100   DISPLAY "USER INTERRUPT RECEIVED"
 55            GO TO 200
 60      150   DISPLAY "NO INTERRUPT"
 65      200   CONTINUE
 70            END
>RUN
 0
 0
NO INTERRUPT

(@70 )>RUN
 0
 ⊕
USER INTERRUPT RECEIVED

(@70 )>RUN
 0
 ☆
-
```

NOTE: The ESC function should be used with caution. If the program should erroneously fall into an infinite loop when ESC(0) is in effect, the only way to terminate execution is to hang up the phone.

The SETSIZE Function

The Tymshare system normally allows a file size no greater than 390,000 characters. This protects the user from accidentally creating an extremely large file. The SETSIZE function allows the SUPER FORTRAN user to set a higher or lower maximum file size if he wishes.

SETSIZE must appear in an assignment statement; its form is

m = SETSIZE(n,u)

where m is a variable of any type, and u is the file unit number. The argument n is the maximum number of elements that the user desires the file to contain, where an element is a character, word, or record, depending on how the file was opened. The value of n may be any numeric expression with a value between 3,840 and 3,248,640, inclusive.

If the system cannot satisfy the request for n elements, the actual number of elements assigned as the file size is returned in the variable m. Normally, m is set to the value n.

For example, to assign a limit of 100,000 characters to the symbolic sequential file on unit 3, the statement is:

m = SETSIZE(100000,3)

The SETSIZE function may be used as often as desired within a single program to change the maximum size of the same file or many different files.

The GOFILE Function

The GOFILE function allows the user to execute any self-standing, user-created GO file[1] from his SUPER FORTRAN program. This function must appear in an assignment statement; its form is

u = GOFILE(v,'file name')

where file name is the name of the self-standing GO file enclosed in single quote marks, and u and v are real variables.

Note that the GOFILE function is designed for use with GO files created from NARP or ARPAS and not subsystem-created GO files.

The running SUPER FORTRAN program is relabeled out, and the self-standing GO file is executed as a fork without subsystem status. The value of the real variable v is passed as a parameter to the GO file, and the real value u is returned. The following conventions must be followed in writing such a self-standing GO file.

The starting location and first-saved address of the GO file must both be greater than 10 decimal. Otherwise, the GO file can use any portion of core it needs. Cells 1, 2, and 3 are used as follows by SUPER FORTRAN to transmit values to the GO file: Cell 1 has the first word of v, cell 2 has the second word of v, and cell 3 has the command file number.

When the GO file is finished, it returns to the SUPER FORTRAN program by executing a BRS 10. The A register should contain the first word of u, the B register should contain the second word of u, and the X register should contain the command file number. Even if no numeric result is appropriate and the A and B registers are thus left undefined, it is very important that the X register be set to the current command file number. SUPER FORTRAN releases all core pages used by the GO file.

1 - Self-standing GO files may be executed with the GO command in the EXECUTIVE.

FEATURES

The CLOSE Statement: Deleting Files

Files may be deleted within a SUPER FORTRAN program by using a form of the CLOSE statement. The form is

CLOSE("file name")

where file name is the name of the file to be deleted.

If a file is open, it must be closed before it can be deleted. This is done with the statement

CLOSE(number)

where number is the number given to the file when it was opened. For example,

```
OPEN(3,"QUAD",RANDIN,SYMBOLIC)
    .
    .
    .
CLOSE(3)
CLOSE("QUAD")
```

opens the file QUAD for symbolic input as a random file. After it has been used, the file is closed and then deleted.

Files need not have been opened before they are deleted with the CLOSE("file name") statement.

If the CLOSE statement cannot be executed, it is ignored, and no error message is printed. This modification applies to both forms of the CLOSE statement. For example,

CLOSE(3)	Closes file 3. If file 3 is not open, the statement is ignored.
CLOSE("DATA")	Deletes the file named DATA. If there is no file DATA in the user's directory, the statement is ignored. However, if DATA is present in the directory, but not deletable, for example, READ ONLY, then a diagnostic is printed.

String Functions and the Null String

If the null string is used as the argument for the VAL and ASC functions, the values 0 and -1 are respectively returned. In other words, VAL(" ") = 0 and ASC(' ') = -1. If the argument of the CHAR function is a negative number, the function returns the null string as its value. Thus, the value of CHAR(-2) is the null string.

Line Feeds

The Line Feed is no longer considered to be a delimiter; thus, it is ignored during data input from the terminal or a file. Therefore, a string variable may not contain a Line Feed.

Error Control

The ERR= function, which may appear in both the free format and formatted forms of the READ and WRITE statements, now permits the user to control additional error conditions. ERR= causes control to transfer to the statement specified if a data transmission error occurs. Additional errors which cause transfer to the specified statement are: VAL(s), where s is a non-numeric string; division by zero; and arithmetic overflow.

When a READ or WRITE statement containing the ERR= function is executed, the system is armed to transfer program control to the statement specified after the equals sign if one of the above errors occurs or if an error occurs during the READ or WRITE process. The system remains armed and transfers control to that statement until:

1. An error occurs which causes control to be transferred to the specified statement.
2. A READ or WRITE statement without an ERR= function is executed.
3. A READ or WRITE statement containing an ERR= function which specifies a different statement is executed. In this case, the system is still armed, but control transfers to the new statement if one of the above errors occurs.

If 1 or 2 above occurs, the system is no longer armed, and all errors are treated in the normal manner. The system remains unarmed until another READ or WRITE statement containing an ERR= function is executed.

For example, when the statement

```
WRITE(1,1,ERR=100)A,B
```

is executed, the system is armed. Then, if any of the above errors occur, control is transferred to statement 100.

This capability can be used without actually performing any file processing by using a dummy WRITE statement. For example,

```
10 WRITE (1,15,ERR=60)
20 15 FORMAT (&)
```

transfers control to statement 60 if one of the above errors occurs.

The MAKEGO Program

The EXECUTIVE MAKEGO program produces a GO file[1] from a SUPER FORTRAN link or binary file. This GO file, together with its associated link files, if any, forms a version-independent system which will continue to run regardless of releases of new versions of SUPER FORTRAN.

1 - GO files may be executed directly from the EXECUTIVE with the command:

-**GO file name**↵

See the *Tymshare EXECUTIVE Reference Manual* for further details.

The MAKEGO program is called from the EXECUTIVE. The form of this command is:

-MAKEGO↵
LINK FILE: binary file name↵
GO FILE: name of GO file to be created↵

MAKEGO then prints the OLD FILE/NEW FILE message and waits for user confirmation.

If the SUPER FORTRAN program consists of only one link, the main program itself, the GO file produced by the MAKEGO program will always run, regardless of the current version of SUPER FORTRAN. If the program consists of more than one link, the GO file and all links must be created in the same version of SUPER FORTRAN. Thus, if the GO file or any of the links is resaved in a new SUPER FORTRAN version, all of the other programs in the set must be resaved as well. Similarly, the link files, though they will function when called from the GO file, will not execute from a later version of SUPER FORTRAN without reloading and resaving.

Example

```
>SAVE TEST↵
TEXT ONLY ?N↵
 OLD FILE↵
OK.
>QUIT↵
```

The user saved his SUPER FORTRAN program on the file TEST. The N(NO) response to the question was necessary to create a binary file.

```
-MAKEGO↵
LINK FILE: TEST↵
GO FILE: GTEST↵
 NEW FILE↵

-
```

The MAKEGO program takes the binary file TEST and produces the SUPER FORTRAN GO file GTEST.

The RENUMBER Command

The SUPER FORTRAN RENUMBER command has been improved. Formerly, with the command

>RENUMBER l_1, l_2 AS l_3(increment)↵

if any renumbered lines would cause original lines to be deleted due to duplicate line numbers, the renumbering was not performed. The new RENUMBER command can renumber all lines without deleting in the case of duplication. This is illustrated below.

The user wishes to renumber the following program with the line numbers 1, 2, 3, and 4.

```
1 DIMENSION A(5)
1.5 ACCEPT A
2 DISPLAY A
2.5 END
```

The previous RENUMBER command would renumber the first line as line 1, but when it tried to renumber line 1.5 as line 2, it recognized that it would have two lines with the line number 2. Thus, to protect the user, it would not perform the renumbering as shown below.

```
>RENUMBER 1:2.5 AS 1(1)↵
NOT ENOUGH ROOM FOR ALL LINES, SOME LEFT ALONE
>LIST↵
  1             DIMENSION A(5)
  1.5           ACCEPT A
  2             DISPLAY A
  2.5           END
>
```

This same command now performs the renumbering.

```
>RENUMBER 1:2.5 AS 1(1)↵
>LIST↵
  1             DIMENSION A(5)
  2             ACCEPT A
  3             DISPLAY A
  4             END
>
```

With all forms of the RENUMBER command, no change is made to the program if a RENUMBER command is given which cannot be completed successfully.

HINTS FOR DEBUGGING LINKS

This section illustrates an easy method for debugging a SUPER FORTRAN program containing links. The user must provide two subroutines which read and write COMMON. They are:

```
SUBROUTINE WCOMMON
COMMON I(COMMON size)
OPEN(3,'scratch file',OUTPUT,BINARY)
WRITE(3)I
CLOSE(3)
END
```

and

```
SUBROUTINE RCOMMON
COMMON I(COMMON size)
OPEN(3,'scratch file',INPUT,BINARY)
READ(3)I
CLOSE(3)
(open any files that remained open in previous program)
END
```

The procedure is described below. First, the user loads his main program into SUPER FORTRAN with the LOAD command. He then gives the MAP command which provides him with the size of COMMON. This is the size used as the dimension of I in the RCOMMON and WCOMMON subroutines. These two subroutines should now be added to the main program and all links.

Next, the user loads his main program, which now contains the two subroutines. He begins execution with the RUN TO command, specifying as breakpoints all of the link statements:

```
>LOAD main program↵
>RUN TO l1,l2,l3,...↵        The li are the link statement references.
```

When the program breaks at the first link statement, the user calls WCOMMON to write the current contents of COMMON on a scratch file:

```
l1 >@CALL WCOMMON↵
```

Then he loads the symbolics of his first link, which contain both subroutines, and gives the INITIALIZE command:

```
l1 >LOAD first link↵
>INITIALIZE↵
```

The INITIALIZE command checks the program structure; if no structural errors are found, it allocates data storage and begins execution of the program, breaking before the first executable statement.

At this point, the user calls RCOMMON to read the current COMMON values into the first link. Then he continues execution, again listing as breakpoints all of the link statements in this link:

```
2 >@CALL RCOMMON↵          In this case, the first executable statement is line 2.
2 >CONTINUE TO l2,l3,...↵
```

This same procedure is continued with each link. Since the LOAD command brings the symbolics of each link program into SUPER FORTRAN (unlike the LINK statement which brings the binary representation into SUPER FORTRAN), the user has complete freedom to debug his program. Note that the RCOMMON subroutine in each link must open any files from the previous link which the user needs open.

Example

The following six-line program is the user's main program.

```
1           COMMON AREA
2           ACCEPT "ENTER BASE AND HEIGHT:",BASE,HEIGHT
3           AREA=BASE*HEIGHT/2
4           DISPLAY "AREA OF TRIANGLE IS:", AREA
5           LINK "VOLUME"
6           END
```

The user loads his program into SUPER FORTRAN and gives the MAP command to determind COMMON size.

```
>LOAD AREA↵
OK.
>MAP↵

6 LINES
TEXT = 147 CHARS (OF 24000)
COMPILATION = 83 BYTES (OF 18000)
NAMES = 5

OBJECT PROGRAM:
SIZE = 45 WORDS
COMMON = 2 WORDS          COMMON size is two words.
DATA STORAGE = 4 WORDS
8117 WORDS UNUSED
```

Now the user can add the two subroutines to his main program and the link. The main program with subroutines is resaved on the file AREA, and the link with subroutines is saved on the file ARBASE. These two files are displayed below.

```
 1          COMMON AREA
 2          ACCEPT "ENTER BASE AND HEIGHT:",BASE,HEIGHT
 3          AREA=BASE*HEIGHT/2
 4          DISPLAY "AREA OF TRIANGLE IS:", AREA
 5          LINK "VOLUME"
 6          END
10          SUBROUTINE WCOMMON
11          COMMON I(2)
12          OPEN(3,"SCOMMON",OUTPUT,BINARY)
13          WRITE(3)I
14          CLOSE (3)
15          END
20          SUBROUTINE RCOMMON
21          COMMON I(2)
22          OPEN(3,"SCOMMON",INPUT,BINARY)
23          READ(3)I
24          CLOSE (3)
25          END
```

```
 1            COMMON ARBASE
 2            ACCEPT "ENTER HEIGHT: ",HEIGHT
 3            VOLUME=ARBASE*HEIGHT
 4            DISPLAY "VOLUME OF REGULAR PRISM IS: ",VOLUME
 5            END
10            SUBROUTINE WCOMMON
11            COMMON I(2)
12            OPEN(3,"SCOMMON",OUTPUT,BINARY)
13            WRITE(3)I
14            CLOSE (3)
15            END
20            SUBROUTINE RCOMMON
21            COMMON I(2)
22            OPEN(3,"SCOMMON",INPUT,BINARY)
23            READ(3)I
24            CLOSE (3)
25            END
```

At this point, the user is ready to run his program and perform any debugging necessary.

```
>LOAD AREA↵          The user loads his main program.
OK.
>RUN TO 5↵                              Line 5 is the only LINK statement.
ENTER BASE AND HEIGHT: 3.1,5.2↵
AREA OF TRIANGLE IS:      8.06

BREAK                          The program breaks at line 5, and the user loads his first link after calling
5 >@CALL WCOMMON↵              WCOMMON to write the current contents of COMMON on a file.

5 >LOAD ARBASE↵
OK.
>INITIALIZE↵          The INITIALIZE command begins execution, breaking at line 2.

2 >@CALL RCOMMON↵          The user calls RCOMMON to read the values of
                           COMMON from the file and continues execution.
2 >CONTINUE↵
ENTER HEIGHT: 9.↵
VOLUME OF REGULAR PRISM IS:      72.54
```

Since the symbolics of the link are in SUPER FORTRAN, the user could now debug the link.

```
(@5 )>LIST↵
  1             COMMON ARBASE
  2             ACCEPT "ENTER HEIGHT: ",HEIGHT
  3             VOLUME=ARBASE*HEIGHT
  4             DISPLAY "VOLUME OF REGULAR PRISM IS: ",VOLUME
  5             END
 10             SUBROUTINE WCOMMON
 11             COMMON I(2)
 12             OPEN(3,"SCOMMON",OUTPUT,BINARY)
 13             WRITE(3)I
 14             CLOSE (3)
 15             END
 20             SUBROUTINE RCOMMON
 21             COMMON I(2)
 22             OPEN(3,"SCOMMON",INPUT,BINARY)
 23             READ(3)I
 24             CLOSE (3)
 25             END
(@5 )>
```

BATCH FORTRAN IV FOR THE SUPER FORTRAN PROGRAMMER

EQUIVALENT NEW FUNCTIONS AND FEATURES

The functions and features described in this section have been added to SUPER FORTRAN to increase its compatibility with BATCH FORTRAN IV. They are, however, available for any SUPER FORTRAN program.

The functions VAL, IVAL, DVAL, and CVAL are all equivalent to the SUPER FORTRAN VAL function. Likewise, the function SFTIME is the same as TIME, and SUBST3 is equivalent to SUBSTR3.

Instead of dimensioning arrays with an asterisk, as in REAL A(*,*), the user may dimension with variables; REAL A(I,J) is thus equivalent to REAL A(*,*). The actual values of I and J are ignored. The method for doing this is described fully on page 22.

If the first non-blank character in a line is a colon (:), that line is considered by SUPER FORTRAN to be a comment. On the other hand, lines beginning with a percent sign (%) as the first non-blank character are compiled; the percent sign is ignored.

THE SUPER FORTRAN COMPATIBILITY MODE OF BATCH FORTRAN IV

The Tymshare BATCH FORTRAN IV and SUPER FORTRAN languages are quite similar syntactically, and programs written in the two languages are generally compatible. The SUPER FORTRAN user who runs one program many times, for instance, a production program, may find it economical and efficient to run his SUPER FORTRAN program in BATCH FORTRAN IV. A special compilation mode, the SUPER FORTRAN compatibility mode, has been added to BATCH FORTRAN IV to minimize the language differences.

This section explains how to run a SUPER FORTRAN program in the SUPER FORTRAN compatibility mode. All existing differences between the languages which would affect the SUPER FORTRAN program are described.

The SUPER FORTRAN compatibility mode is called by giving the SFORTRAN command at BATCH FORTRAN IV command level prior to compilation.

The SUPER FORTRAN compatibility mode can compile SUPER FORTRAN programs with line numbers. If the SUPER FORTRAN program does not include line numbers, the user should give the OFF LINE NUMBERS command after he has given the SFORTRAN command in BATCH FORTRAN IV. This compatibility mode accepts files created with the SUPER FORTRAN SAVE command whether the TEXT ONLY? question is answered with a Y or N.

Example

The following example illustrates the ease with which SUPER FORTRAN programs can be run in BATCH FORTRAN IV.

The user lists and runs his SUPER FORTRAN program in SUPER FORTRAN.

```
-SFO↵

>LOAD BAL↵
OK.
>LIST↵
 10              C: DOUBLE DECLINING BALANCE DEPRECIATION PROGRAM
 20              WRITE(1,2) "COST OF ASSET= $"
 30       2      FORMAT(//S,&)
 40              READ(0,3)C
 50       3      FORMAT(F12.2)
 60              WRITE(1,4) 'ESTIMATED USEFUL LIFETIME= '
 70       4      FORMAT(S,&)
 80              READ(0,3)U
 90              WRITE(1,5) 'YEAR','DEPRECIATION','BOOK VALUE'
100       5      FORMAT(/S,8X,S,8X,S)
110              DO 100 I=1,U
120              D=2*C/U
130              C=C-D
140              WRITE(1,7)I,D,C
150       7      FORMAT(I4,8X,'$',F10.2,8X,'$',F8.2)
160       100    CONTINUE
170              END
>RUN↵

COST OF ASSET= $3500.00↵
ESTIMATED USEFUL LIFETIME= 7.↵

YEAR          DEPRECIATION          BOOK VALUE
   1          $    1000.00         $ 2500.00
   2          $     714.29         $ 1785.71
   3          $     510.20         $ 1275.51
   4          $     364.43         $  911.08
   5          $     260.31         $  650.77
   6          $     185.93         $  464.84
   7          $     132.81         $  332.03

(@170 )>QUIT↵
```

```
-BFO↵
```
The user calls BATCH FORTRAN IV and gives the SFORTRAN command to compile in the compatibility mode.

```
+SFORTRAN↵

+COMPILE BAL,BF4BAL↵
 NEW FILE↵
```
His program was saved on the file BAL; the compiled version is saved on BF4BAL.

```
1 70 END

+LOAD BF4BAL↵
```
He loads his object program.

```
+WRITE BAL4↵
LOADING LIBRARY
 NEW FILE↵
```
The user creates a GO file with the WRITE command.

```
+RUN↵
```
The user begins execution with RUN.

```
COST OF ASSET= $3500.00↵
ESTIMATED USEFUL LIFETIME= 7.↵

YEAR          DEPRECIATION          BOOK VALUE
   1          $    1000.00         $ 2500.00
   2          $     714.29         $ 1785.71
   3          $     510.20         $ 1275.51
   4          $     364.43         $  911.08
   5          $     260.31         $  650.77
   6          $     185.93         $  464.84
   7          $     132.81         $  332.03

+QUIT↵
```

For future runs, the user only needs to execute the GO file BAL4 directly from the EXECUTIVE; he does not need to recompile the program.

Described below are the remaining characteristics of BATCH FORTRAN IV in the SUPER FORTRAN compatibility mode which differ from SUPER FORTRAN.

1. Only the first six rather than the first 31 characters of an identifier name are used to distinguish between identifiers in the SUPER FORTRAN compatibility mode. In addition, only upper case letters may be used in identifiers.
2. Brackets cannot be used in place of parentheses. A SUPER FORTRAN program using brackets will not compile in the compatibility mode.
3. The operators .EQU. and .IMP. are not recognized in the compatibility mode, and programs using them will not compile.
4. The SUPER FORTRAN functions SIGNUM, ENTIER, ROUND, POLAR, FACT, TIME, TEL, WAIT, TCP, ESC, SETSIZE, GOFILE, ONLC, YEAR, and EXEC are not included in the libraries. These functions will be listed as missing when a program calling them is loaded. The TIME function is included in the compatibility mode with the name SFTIME.

5. Programs with logicals equivalenced to non-logicals in an EQUIVALENCE statement do not work in the compatibility mode as they do in SUPER FORTRAN since the conversion from .TRUE. to 1 and .FALSE. to 0 does not occur. BATCH FORTRAN IV converts .TRUE. to -1 and .FALSE. to 0. Assignments between logical and numeric variables are compatible, however.
6. Logical constants in a DATA statement must be either .TRUE. or .FALSE.. String constants in DATA statements must be enclosed in quote marks.
7. Logical arrays are stored one element per word rather than 24 elements per word. Thus, care should be taken if logical arrays are being equivalenced to non-logical variables.
8. Complex input/output using the A conversion requires two specifications rather than one per variable or expression.
9. Labels range from 1 to 99999 rather than 0 to 99999.
10. Type and dimension declarations must appear before the first use of the identifier.
11. A program containing the largest of a specific COMMON block must be loaded before all other programs containing that COMMON block. Otherwise, when the program is loaded, SUPER FORTRAN prints the name of the COMMON block which is larger than a previously loaded copy, followed by an error message.
12. BATCH FORTRAN IV subroutines and machine language subroutines can be used with a SUPER FORTRAN program executed in the SUPER FORTRAN compatibility mode.
13. The EXTERNAL statement is required to recognize subroutine names as subroutine arguments.
14. Statement functions must appear before any executable or FORMAT statement.
15. Calling sequence arguments must agree in type with the dummy arguments of the subroutine called. Using an asterisk array to pass array dimensions is not recognized or diagnosed. If the SUPER FORTRAN user wishes to pass array bounds, the following changes can be made to his program; it will then be acceptable in both SUPER FORTRAN and the SUPER FORTRAN compatibility mode.

    a. In the call, for each subscript that is to be passed, one integer variable is added to the end of the calling sequence. Before the call, each of these variables is set to the corresponding dimension.

    b. In the receiving sequence, integer variables are added correspondingly to the ones in the call. The asterisk in the dimension of the dummy arrays is replaced with the appropriate integer variable.

    For example,

Old Program

```
DIMENSION A(10,4),B(6)
     .
     .
     .
CALL SUB(X,A,Y,B)
     .
     .
     .
SUBROUTINE SUB(T,D,V,E)
DIMENSION D(*,*),E(*)
     .
     .
     .
```

New Program

```
DIMENSION A(10,4),B(6)
     .
     .
     .
IA1=10;IA2=4
CALL SUB(X,A,Y,B,IA1,IA2,6)
     .
     .
     .
SUBROUTINE SUB(T,D,V,E,ID1,ID2,IE1)
DIMENSION D(ID1,ID2),E(IE1)
     .
     .
     .
```

16. In SUPER FORTRAN, string, logical, double precision, and complex functions need not and can not be declared in other subroutines or in the main program in which they are used. BATCH FORTRAN IV requires the string functions to be declared. To handle this situation, the following conventions have been added to SUPER FORTRAN and to the SUPER FORTRAN compatibility mode.

SUPER FORTRAN	If the first non-blank character in a line is a percent sign (%), it is ignored, and the rest of the line is compiled. If the first non-blank character in a line is a colon (:), the entire line is ignored.
SUPER FORTRAN Compatibility Mode	If the first non-blank character in a line is a percent sign (%), the entire line is ignored. If the first non-blank character in a line is a colon (:), it is ignored, and the rest of the line is compiled.

The string, logical, double precision, or complex declarations can then be included in the SUPER FORTRAN program by preceding the statement with a colon. The declarations will be ignored in SUPER FORTRAN but compiled in the SUPER FORTRAN compatibility mode.

Example

```
10 :STRING REV(3)
11 EXTERNAL REV
20 STRING S(3)
30 S=REV("ABC")
40 END
50 STRING FUNCTION REV(T)(3)
60 STRING T(3)
70 REV=RIGHT(T,1)+SUBSTR3(T,2,1)+LEFT(T,1)
80 RETURN
90 END
```

In addition, the percent sign feature allows the inclusion of debugging statements in the program while running in SUPER FORTRAN which are ignored when the program is executed in BATCH FORTRAN IV.

Execution of SUPER FORTRAN programs and their links in the SUPER FORTRAN compatibility mode is identical to their execution in SUPER FORTRAN. However, the preparation of link files and the debugging of link programs is quite different from SUPER FORTRAN. Note that this difference does not concern the user who debugs his links in SUPER FORTRAN before moving them to BATCH FORTRAN IV.

BATCH FORTRAN IV only runs one SUPER FORTRAN program at a time. If a program executes a LINK statement, the program is destroyed. COMMON is preserved, and the new link is run. Any files open in the calling program remain open when the new link is run. However, a maximum of three files may be open when a LINK statement is executed.

Link files are produced by use of the WRITE command; GO files are produced by the DUMP command. The user may not link to files created with the DUMP command. There is no LINK command in BATCH FORTRAN IV. The LINK command is accepted, however, in the SUPER FORTRAN compatibility mode.

For example, consider the following SUPER FORTRAN program with line numbers removed for BATCH FORTRAN IV. In BATCH FORTRAN IV, this is run as follows. Assume the symbolic files are called SYMAREA and SYMVOLUME.

Example

Suppose the file SYMAREA contains the program

```
1           COMMON AREA
2           ACCEPT "ENTER BASE AND HEIGHT:",BASE,HEIGHT
3           AREA=BASE*HEIGHT/2
4           DISPLAY "AREA OF TRIANGLE IS:", AREA
5           LINK "VOLUME"
6           END
```

and the file SYMVOLUME contains the program whose symbolics are:

```
1           COMMON ARBASE
2           ACCEPT "ENTER HEIGHT: ",HEIGHT
3           VOLUME=ARBASE*HEIGHT
4           DISPLAY "VOLUME OF REGULAR PRISM IS: ",VOLUME
5           END
```

The binary file for VOLUME is compiled, loaded, and prepared:

```
-BFORTRAN4↵

+SFORTRAN↵

+COMPILE SYMVOLUME,BINVOLUME↵
 NEW FILE↵

5 END
```

```
+LOAD BINVOLUME↵

+WRITE VOLUME↵
LOADING LIBRARY
 NEW FILE↵
```

SYMAREA is compiled and loaded:

```
+COMPILE SYMAREA,BINAREA↵
CLEAR ALL OVERLAYS? YES↵
 NEW FILE↵

6 END

+LOAD BINAREA↵

+RUN↵
LOADING LIBRARY
ENTER BASE AND HEIGHT:3.1,5.2↵
AREA OF TRIANGLE IS:     8.06
ENTER HEIGHT: 9↵
VOLUME OF REGULAR PRISM IS:      72.54

+
```

At this point, unlike SUPER FORTRAN, the program AREA, which was initially run, is completely forgotten, and the new link VOLUME is in core with complete BATCH FORTRAN IV debugging capabilities available. A second RUN command reruns the VOLUME program, not the AREA program.

If the user wishes to create a GO file of the AREA program to be run in the EXECUTIVE, he gives the DUMP command after the file BINAREA is loaded. For example,

```
+LOAD BINAREA↵

+DUMP TEST↵
LOADING LIBRARY
 NEW FILE↵

+QUIT↵

-GO TEST↵
ENTER BASE AND HEIGHT:3.2,5.2↵
AREA OF TRIANGLE IS:     8.32
ENTER HEIGHT: 9↵
VOLUME OF REGULAR PRISM IS:      74.88

-
```