# FORTRAN 77

The Programmer's Companion

PRIME
Computer

**The Programmer s Companion** is a new series of pocket size quick reference guide to Prime Software products

Written and published by Prime Computer Technical Publications Department 500 Old Connecticut Path Framingham MA 01701 telephone (617) 879 2960 8 30 5 00 PM Eastern Time



This document reflects the software as of Master Disk Revision 17

Printing history
First printing August 1980 15M

Credits

**Research and copy**
John Mann

**Design and production**
William Agush

**Typesetting**
JI Associates

**Cover design**
William Agush

**Printing and binding**
Mark-Burton

**Paper**
S D Warren (Cover)
Finch Paper (Text)

# TABLE OF CONTENTS

# TYPOGRAPHIC CONVENTIONS

The following conventions are used in this Programmer's Companion

| | |
|---|---|
| **Braces { }** | Braces indicate a choice of options or arguments Unless the braces are enclosed by brackets one choice must be selected |
| **Brackets [ ]** | Brackets indicate that the item enclosed is optional |
| **Ellipsis .** | An ellipsis indicates that the preceding item may be repeated |
| **Parentheses ( )** | When parentheses appear in a statement format they must be included literally when the statement is used |
| **WORDS-IN-UPPER-CASE** | Uppercase letters identify command words or keywords They are to be entered literally |
| **words-in-lower-case** | Lowercase letters identify options or arguments The user substitutes an appropriate numerical or text value |

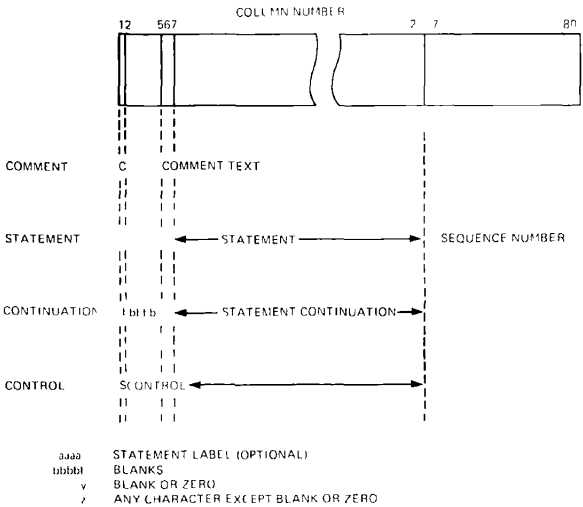
# LEGAL CHARACTER SET

Any ASCII character may appear in FORTRAN 77 character data Hollerith constants and I/O files In program source statements the legal characters are

- The 26 uppercase letters A-Z
- The 26 lowercase letters a-z
- The 10 digits 0-9
- The 13 special characters = + - * / ( ) $ _ (or ←)
- Blanks or spaces

# LINE FORMAT

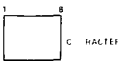
Each program line is a string of 1 to 72 characters. Each character position in the line is called a column, numbered from left to right starting with 1. The following is a schematic of a program line:




Note: bbbbb may be a statement number but control cannot be transferred to it.

# DATA TYPES

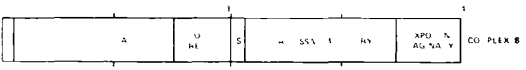
## CHARACTER



A sequence of bytes, each holding one ASCII character

| | |
|---|---|
| **Range·** | 1 to 32767 characters |
| **Constant** | cccccc' Represent an internal single quote with two consecutive single quotes |

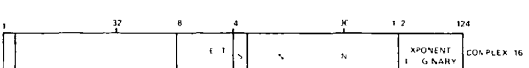
## COMPLEX (COMPLEX*8)



Two REAL*4 numbers representing the real and imaginary parts

| | |
|---|---|
| **Bytes·** | 4 + 4 |
| **Range.** | Each component has same range as REAL |
| **Constant:** | (Real_part Imaginary part) In formatted I/O the parentheses and comma are omitted |

## COMPLEX*16



Two DOUBLE PRECISION numbers representing the real and imaginary parts

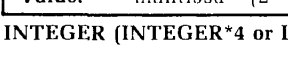
| | |
|---|---|
| **Bytes·** | 8 + 8 |
| **Range·** | Each component has same range as DOUBLE PRECISION |
| **Constant.** | (Real_part Imaginary_part) In formatted I/O, the parentheses and comma are omitted |

## DOUBLE PRECISION (REAL*8)


A real number in double precision form.

| | |
|---|---|
| **Bytes:** | 8 |
| **Range:** | ± (10 **-9902 to 10**9825) |
| **Constant:** | [ ±] mantissa D[ ±] exponent. The mantissa optionally may contain a decimal point. |
| **Precision:** | 47 bits or 14 decimal digits |
| **Value:** | mantissa * (2**(exponent)) |

## INTEGER (INTEGER*4 or Long Integer)

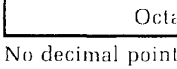



An integer in twos-complement form.

| | |
|---|---|
| **Bytes:** | 4 |
| **Range:** | -(2**31) to (2**31-1) |
| | Decimal -2147483648 to 2147483647 |
| | Octal :0 to :37777777777 |
| **Constant:** | Decimal [ ±] ddddd |
| | Octal [ ±]:ddddd |

No decimal point may appear in an INTEGER data item.

## INTEGER*2 (Short integer)

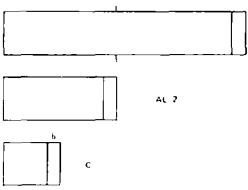



An integer in twos-complement form.

| | |
|---|---|
| **Bytes:** | 2 |
| **Range:** | -(2**15) to (2**15 1) |
| | Decimal -32768 to 32767 |
| | Octal 0 to 177777 |
| **Constant:** | Decimal [ ±] ddddd |
| | Octal [ ±] ddddd |

No decimal point may appear in an INTEGER*2 data item

**LOGICAL**



Hold a logical value All bits are zero except the last which may be zero (false) or one (true)

| | |
|---|---|
| **Bytes·** | 1 2 or 4 |
| **Range** | True or false |
| **Constants:** | TRUE (or T in an input file) |
| | FALSE (or F in an input file) |

LOGICAL*1 is provided only as an aid to program conversion

**REAL(REAL*4)**

Holds a real number in single precision form

| | |
|---|---|
| **Bytes:** | 4 |
| **Range** | ± (10 **-38 to 10**38) |
| **Constant:** | [ ±] mantissa [E[ ±] exponent] |
| | The mantissa must contain a decimal point if the exponent is omitted but otherwise need not |
| **Precision:** | 23 bits or 6 decimal digits |
| **Value·** | Mantissa * (2**(exponent)) |

# STATEMENT LABEL

A statement label is an integer constant that is prefixed to a statement A label may appear anywhere in columns 1 5

| | |
|---|---|
| **Range** | 1 to 99999 |
| **Constant** | $label or *label (Statement label constants are used in alternate returns from sub routines ) |

# OPERANDS

### Arrays

An array is an ordered possibly multidimensional set of variables An array is declared in a DIMENSION COMMON or type statement such as

**DIMENSION array declarator [,array declarator]**

where each **array declarator** has the form

**ANAME (d1[ d2] [,d7])**

in which **ANAME** is the name the array is to have (same rules as for a variable name) and each **dn** has the form

***[Ln ]Hn***

**Ln** is the lower subscript bound and **Hn** is the upper subscript bound for dimension **n** There may be at most seven dimensions If **Ln** is omitted it is assumed to be 1

FORTRAN 77 arrays are stored by columns the leftmost subscript varies most rapidly when the array is accessed in storage order

### Constants

A constant is a literal representation of a value The correct form for a constant of each data type is shown above in the description of the type

### Parameters

In FORTRAN 77 a parameter is a named constant not an element in the argument list of a subprogram entry point A parameter is declared in a PARAMETER statement and may be used wherever a constant could be used except in a FORMAT statement Parameter names follow the same rules as variable names

### Variables

A variable is a data item whose value may be assigned during program execution Variable names may contain from 1 to 32 characters Character 1 must be alphabetic the rest must be alphanumeric $ or * Users are discouraged from using $ in their variable names because this character is used extensively in Prime supplied software names where it serves to implement a system of naming conventions

When no type is explicitly declared a variable whose name begins with the letters I through N becomes type INTEGER and a variable whose name begins with A H or O Z becomes type REAL This convention can be overridden by a type statement or an IMPLICIT statement

# OPERATORS

### Logical operators

**NOT**
**AND**
**OR** (non exclusive)
**EQV**
**NEQV**

NOT AND and OR are generally known The truth tables for EQV and NEQV are

**EQV** (P EQV Q) is the logical equivalence of P and Q

| | P | |
|---|---|---|
| Q | TRUE | FALSE |
| TRUE | TRUE | FALSE |
| FALSE | FALSE | TRUE |

**NEQV** (P NEQV Q) is the same in effect as ( NOT (P EQV Q)) It acts as an exclusive or

| | P | |
|---|---|---|
| Q | TRUE | FALSE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | FALSE |

### Arithmetic operators

| | |
|---|---|
| ** | Exponentiation |
| * | Multiplication |
| / | Division |
| + | Addition |
| - | Subtraction or Unary Minus |
| = | Assignment |

### Relational operators

| | |
|---|---|
| LT | Less than |
| LE | Less than or equal to |
| EQ | Equal to |
| NE | Not equal to |
| GT | Greater than |
| GE | Greater than or equal to |

### Character operator

| | |
|---|---|
| // | Concatenation |

### Operator priority

| | |
|---|---|
| ** | Exponentiation |
| - | Unary Minus |
| * or / | Multiplication or division |
| + or - | Addition or subtraction |
| // | Concatenation |
| LT LE EQ NE GT GE | Relational operators (All have same priority) |
| NOT | Logical negation |
| AND | Logical intersection |
| OR | Logical union |
| EQV NEQV | Logical equivalence/ nonequivalence |

### Order of evaluation

Operators of higher priority are evaluated before operators of lower priority. Operators of equal priority are evaluated right to left in the case of multiple exponentiation and left to right otherwise. Expressions within parentheses are evaluated before operations outside the parentheses are performed.

The F77 compiler sometimes re-arranges mathematical expressions into equivalent forms which can be evaluated more quickly. When this occurs, evaluation order may not be strictly as described. However the compiler always respects the integrity of parentheses. Where evaluation order is critical, use parentheses to specify it uniquely.

Function references may be evaluated in any order The order used cannot be specified by the programmer

# TYPE CONVERSION

### Arithmetic conversion

The type of the result when differing numeric types are combined will be that of the operand having the higher type in the following list

COMPLEX*16
COMPLEX*8
DOUBLE PRECISION
REAL
LONG INTEGER
SHORT INTEGER

For Example REAL + SHORT INTEGER is a REAL

**Special case:** To prevent loss of precision the result type when COMPLEX*8 and DOUBLE PRECISION data are combined will be COMPLEX*16

**Caution**

When long integers are converted to reals there may be a loss of precision

### Character conversion

When one character item is assigned to another and their lengths are not the same padding or truncation takes place on the right

### Logical conversion

The storage length of the result when logical data of differing lengths are combined is the longer of the two lengths

# PROGRAM COMPOSITION

Each program unit consists of a number of program lines Program lines are grouped and ordered as shown in the following table Vertical boundaries in the table denote classes of statements that can be interspersed Horizontal boundaries denote classes of statements that cannot be interspersed

Any number of program units may be present in a single file. Only comments may appear between the END statement of one program unit and the header statement of the next.

In F77 no block of executable code can cross a segment boundary. Therefore no program unit may produce more than 128K bytes (one segment) of code. Rarely if ever will a program unit be any larger than this; one that is must be broken up. The local data for an F77 program unit is kept in its stack frame (dynamic data) and link frame (static data). Neither of these frames may be larger than a segment. One that is must be reduced in size by moving some of its data to COMMON.

The names of F77 program units may not be more than 8 characters long. Additional characters will be ignored and a warning message printed.

# PROGRAM SPECIFICATION STATEMENTS

## ▶ Assignment Statements

**target - expression**

An assignment statement evaluates the **expression** and assigns its value to the **target**. Where type conversion is required it occurs automatically.

The value of a logical expression will be converted when necessary to the storage length of the target. The value of a character expression will be padded or truncated on the right when necessary to match the length of the target.

An arithmetic target can be assigned a value of any arithmetic type. The following table gives the results of mixed type arithmetic assignments.

| | Target Type | | | | | |
|---|---|---|---|---|---|---|
| **Value Type** | **I*2** | **I*4** | **REAL** | **DOUBLE** | **C*8** | **C*16** |
| **I*2** | ASSIGN | EXTEND<br>ASSIGN | FLOAT<br>ASSIGN | DFLOAT<br>ASSIGN | FLOAT<br>ASREAL | DFLOAT<br>ASREAL |
| **I*4** | TRUNC<br>ASSIGN | ASSIGN | FLOAT<br>ASSIGN | DFLOAT<br>ASSIGN | FLOAT<br>ASREAL | DFLOAT<br>ASREAL |
| **REAL** | SFIX<br>ASSIGN | LFIX<br>ASSIGN | ASSIGN | DFLOAT<br>ASSIGN | ASREAL | DFLOAT<br>ASREAL |
| **DOUBLE** | SFIX<br>ASSIGN | LFIX<br>ASSIGN | FLOAT<br>ASSIGN | ASSIGN | FLOAT<br>ASREAL | ASREAL |
| **C*8** | SFIX*<br>ASSIGN* | LFIX*<br>ASSIGN* | ASSIGN* | DFLOAT*<br>ASSIGN* | ASSIGN | DFLOAT<br>ASSIGN |
| **C*16** | SFIX*<br>ASSIGN | LFIX*<br>ASSIGN* | FLOAT*<br>ASSIGN* | ASSIGN | FLOAT<br>ASSIGN | ASSIGN |

| Operation | Action |
|---|---|
| **ASSIGN** | Transmit value (after any indicated conversion) to the target |
| **ASREAL:** | ASSIGN value as above to the real part of a complex number and set the imaginary part of the complex number to zero |
| **SFIX** | Discard fraction leaving a whole number. Convert result to a short integer. Overflow may occur |
| **LFIX** | Discard fraction leaving a whole number. Convert result to a long integer. Overflow may occur |

**FLOAT:** Convert value to REAL form. Loss of precision may occur if the argument was DOUBLE PRECISION, COMPLEX*16 or INTEGER*4. Overflow may occur with DOUBLE PRECISION or COMPLEX*16.

**DFLOAT:** Convert value to DOUBLE PRECISION form.

**EXTEND:** Prefix the short integer with 16 binary 0's or 1's if the short integer was positive or negative respectively. This cannot change the value or sign of the integer.

**TRUNC:** Discard the 16 high-order bits of the long integer. A value outside the short-integer range will be altered, and possibly changed in sign, by this operation.

An asterisk affixed to an operation involving a complex number indicates that the operation is to be performed on the real part *only* — the imaginary part is not involved. When no asterisk is present, the operation is to be performed on both parts of the number.

**▶ASSIGN k TO i**

Assigns a statement number **k** to the integer variable **i**.

**▶BLOCK DATA [name]**

Header statement for a block data subprogram.

**▶CALL subroutine [( [argument [,argument]...] )]**

Calls the specified **subroutine** with an optional list of **arguments.**

**▶COMMON /x/a [,/x/a]... (Comma is optional)**

Defines COMMON blocks. Each **x** is a COMMON block name (named COMMON) or is omitted, leaving two adjacent slashes (blank COMMON). Each **a** is a list of data elements. A COMMON block name may have at most eight characters.

**▶[Label] CONTINUE**

The CONTINUE statement does nothing. Control proceeds to the next statement to be executed.

**▶DATA k/d/ [,k d ] (Comma is optional)**

Initializes variables or array elements **k** to the values **d** at load time

**▶DIMENSION array declarator [,array declarator]..**

Each **array declarator** is as described under ARRAYS (above)

A DIMENSION statement declares a symbolic name typed in a type statement or by default to be an array and sets the number of dimensions and the bounds of each dimension of the array

A list of arrays can be declared and typed in one statement by replacing the keyword DIMENSION above with any data type specifier

**▶DO label [,] i m1 m2, [ m3]**

| | |
|---|---|
| **label** | The statement number of the last statement (usually a CONTINUE) to be executed by the DO-loop |
| **i:** | An INTEGER REAL or DOUBLE PRECISION variable used as the index |
| **m1, m2, m3** | INTEGER REAL or DOUBLE PRECISION *expressions representing* the initial limit and increment values respectively for the index. The default for *m3* is one |

The FORTRAN 77 DO loop differs in many ways from the FORTRAN IV DO-loop Programmers not completely familiar with the FORTRAN 77 version should consult the FORTRAN 77 Reference Guide

**▶END**

The final statement of a program subroutine (including a BLOCK DATA subroutine) or external function Tells the compiler that it has reached the physical end of the program unit

**▶ENTRY name [( [argument [,argument] .] )]**

Specifies a secondary entry point in a subprogram assigns its **name**, and specifies its dummy **arguments**

► **EQUIVALENCE (k ,k [,k]... ) [,(k ,k [,k]... )] . . .**

Causes all the items mentioned in each parenthesized list to be stored beginning with the same byte of physical storage When variables of different lengths are equivalenced the shorter is stored in the first bytes of the longer When specific array elements are equivalenced, the arrays as wholes become correspondingly aligned

► **EXTERNAL subprogram [,subprogram]...**

Allows the subprograms specified to be passed as arguments to other subprograms, where they may be used directly or declared EXTERNAL and passed again

► **[type] FUNCTION name ( [argument [,argument]. .] )**

Declares a program unit to be a function assigns its **name** and **type,** and specifies its dummy **arguments.** If no **type** is declared in the FUNCTION statement, the typing can be done in an ordinary type-statement If no **type** is declared anywhere default typing will occur

► **GO TO i [[,] (k [,k] ..)]**

(Assigned ) The i is an integer variable and each **k** is the label of an executable statement in the program unit containing the assigned GO TO Prior to executing the assigned GO TO a statement label value must be assigned to i using the ASSIGN statement Transfers control to the statement whose label was assigned to i

► **GO TO (k [,k] ) [,] i**

(Computed ) Transfers control to the statement whose label is in the n th position in the list of k's when integer expression i=n If there is no n'th statement label control passes to the next executable statement after the computed GO TO

► **GO TO k**

(Unconditional ) Transfers control to statement **k.**

► **IF (e) statement**

(Logical ) The e is a logical expression and **statement** is any statement except a DO Logical-IF block-IF, ELSE IF, ELSE or END IF statement If **e** is true, the **statement** is executed if e is false control passes to the next executable statement

►**IF (e) THEN**
**[statements]**
**[ELSE IF (e) THEN]**
**[statements]**
**[ELSE]**
**[statements]**
**END IF**

(Block) Allows a block of statements to be executed if an associated logical expression **e** is true or skipped if it is false. Scans a series of such blocks, executes the first whose expression is true and skips over the remaining blocks automatically.

There may be any number of ELSE IF statements or none. There may be at most one ELSE statement, which must follow any ELSE IF statements. The blocks may contain any number of statements or none.

►**IMPLICIT type (list) [,type (list)]..**

Allows the programmer to override the language convention for default data-typing by first letter. Each **type** is a data type such as REAL*4, COMPLEX etc. Each **list** lists the letters which will cause default to that type. Letters may be separated by a comma or an inclusive group of letters may be indicated with a dash.

►**$INSERT insert-file**

Inserts into the program at compilation time the file whose pathname is **insert-file.** The $INSERT command cannot be nested. It must begin in column 1.

►**INTRINSIC name [,name]**

Each **name** is the name of an F77 intrinsic (built in) function. Allows the functions listed to be passed as arguments to subprograms, which may then reference the particular function passed.

►**LIST**

Reverses the effect of a *NO LIST* statement. Source-listing generation resumes (or begins) following the LIST statement. The LIST statement does not of itself cause source listing to be generated.

### ▶NO LIST

If a source listing of any kind has been specified in the compiler options a NO LIST statement will suppress generation of the listing for source lines following the statement. Otherwise NO LIST has no effect.

### ▶PARAMETER (p=c [,p c] ) (Parentheses optional)

The **p**s are symbolic names previously typed in any standard way or by default. Each **c** is a constant expression of a type appropriate to the corresponding **p**. A PARAMETER statement declares each **p** to be a parameter (a named constant) having the value given by **c**.

### ▶PAUSE [n]

**n** is an optional decimal number of up to five digits, or a character constant. Halts the program and prints ****PAUSE **n** at the terminal. Typing in the command START causes execution of the program to resume at the next executible statement following the PAUSE.

### ▶PROGRAM name

Gives a **name** to a main program. This statement is not required. If present it must be the first statement of the main program.

### ▶RETURN [n]

Used in a subprogram to cause return to the calling program unit. Any number of RETURN statements may appear. In a subroutine the integer expression **n** may be specified. Execution of RETURN **n** causes return to the statement of the calling program unit whose label was passed as the n'th statement-label dummy argument in the subroutine argument list. If there is no such argument, a normal return occurs.

### ▶SAVE [v[,v] ]

Causes the subprogram variables and arrays named in it to retain their values between invocations (static storage) rather than losing their values when the subprogram returns (dynamic storage). If no **v**s appear the SAVE is taken to include all local data items.

**▶STOP [n]**

The **n** is an optional decimal number of up to five digits or character constant. Halts program execution, closes all file units referenced by the program, prints ****STOP n** at the terminal and returns control to the PRIMOS level. A STOP statement may appear anywhere in a program unit.

**▶SUBROUTINE name [( [argument [,argument] ] )]**

Declares a program unit to be a subroutine, assigns its **name** and specifies its dummy **arguments**.

**▶type k[/d/] [,k[d]]**

Each **k** is a data item name, each **d** is a value (or list of values for an array) which if present will be used to initialize the corresponding **k**. Allows override of the implicit type assignments of symbol names which would otherwise be done by an IMPLICIT statement or by default.

# INPUT/OUTPUT STATEMENTS

FORTRAN and PRIMOS use different numbering conventions for the set of file units. Beware of confusing the two systems.

| FORTRAN unit-number | PRIMOS device |
|---|---|
| 1 | User terminal |
| 2 | Paper tape reader punch |
| 3 | Parallel interface card reader |
| 4 | Serial line printer |
| 5, 20 | Funit 1-16 |
| 21-24 | 9 track magnetic tape unit 0-3 |
| 25-28 | 7 track magnetic tape unit 0-3 |
| 29-130 | Funit 17-127 |

**▶BACKSPACE ([UNIT ]unit# [ IOSTAT= ios] [ ERR label])**

The options are as described for the OPEN statement.

Moves the pointer of a file open for sequential access back to the beginning of the previous record. Cannot be used on unformatted varying length records or on records written using list directed I/O.

### ▶CLOSE ([UNIT= ]unit# [,STATUS stat] [,ERR= label] [,IOSTAT= ios])

The CLOSE statement disconnects a file from a unit. **ERR=** and **IOSTAT=** have the same significances as in the OPEN statement. **STATUS=** determines the final disposition of the file. The argument **stat** is a character expression which may have the values:

| | |
|---|---|
| **'KEEP'** | The file will be retained after it is closed. This is the default for non SCRATCH files and must not be given for SCRATCH files. |
| **'DELETE'** | The file will be deleted after it is closed. Default for SCRATCH files. |

The options used may be given in any order, except that if **UNIT=** is omitted, **unit#** must appear first.

### ▶ENDFILE ([UNIT= ]unit# [,IOSTAT= ios] [,ERR= label])

The options are as described for the OPEN statement.

Writes a device specific endfile record on the file connected to the file unit **unit#**. The pointer is left positioned after the endfile record. This statement can also be used to truncate a file.

On a DAM file, no endfile record should ever be written.

### ▶Label FORMAT (format)

The FORMAT statement provides one way to specify a format for a READ, WRITE or PRINT. Formats are described below, under FORMATS.

### ▶INQUIRE statement

```
INQUIRE     ([FILE=]filename or [UNIT=]unit# [IOSTAT=ios]
            [ERR=s] [EXIST=ex] [OPENED=od] [NUMBER=num]
            [NAMED=nmd] [NAME=fn] [ACCESS=acc]
            [SEQUENTIAL=seq] [DIRECT=dir] [FORM=fm]
            [FORMATTED=fmt] [UNFORMATTED=unf] [RECL=rcl]
            [NEXTREC=nr] [BLANK=blnk])
```

Used to ascertain the properties of a file or of its connection to a unit.

The file must be specified by **name** (INQUIRE by name) or **unit** (INQUIRE by unit) but not both. Options may appear in any order, but no option may appear more than once. If **FILE=** (or **UNIT=**) is omitted, the **filename** (or **unit#**) must appear first.

The INQUIRE statement options are defined as follows:

| Specifier | Argument Data Type | Significance of Possible Values |
|---|---|---|
| **FILE** | Character Expression | Specifies file by name |
| **UNIT** | Integer*4 Expression | Specifies file by unit number |
| **IOSTAT** | Integer*4 | **Zero** no error condition exists |
| | | **Positive** error condition exists |
| **ERR** | Statement number | Control transfers to statement indicated if error occurs during INQUIRE statement execution |
| **EXIST** | Logical*4 | **TRUE** the file exists (for INQUIRE by name) or the unit exists (for INQUIRE by unit) |
| | | **FALSE** the file or the unit does not exist |
| **OPENED** | Logical*4 | **TRUE** the file is open (INQUIRE by name) or the file unit is open (INQUIRE by unit) |
| | | **FALSE** the file or the unit is not open |
| **NUMBER** | Integer*4 | Variable supplied is set to the file's unit number. If there is none, variable becomes undefined |
| **NAMED** | Logical*4 | **TRUE** the file has a name |
| | | **FALSE** the unit has no name |
| **NAME** | Character | Variable is set to the file name. If none (file not connected) variable becomes undefined |
| **ACCESS** | Character | **SEQUENTIAL** file open for sequential access |
| | | **DIRECT** file open for direct access |
| | | Becomes undefined if file is closed |
| **SEQUENTIAL** | Character | **YES** file can be connected for sequential access |
| | | **NO** file cannot be connected for sequential access |
| | | **UNKNOWN** suitability of the file for sequential access cannot be determined |
| **DIRECT** | Character | **YES** file can be connected for direct access |
| | | **NO'** file cannot be connected for direct access |
| | | **UNKNOWN** suitability of file for direct access cannot be determined |

| | | |
|---|---|---|
| **FORM** | Character | **'FORMATTED** open for formatted data transfer<br>**'UNFORMATTED** open for unformatted data transfer<br>Becomes undefined if file is not open |
| **FORMATTED** | Character | **'YES** file consists of formatted records<br>**'NO** file consists of unformatted records<br>**UNKNOWN** record type cannot be determined |
| **UNFORMATTED** | Character | **'YES** file consists of unformatted records<br>**'NO** file consists of formatted record<br>**UNKNOWN** record type cannot be determined |
| **RECL=** | Integer*4 | Variable is set to the record length for which the file is open. Becomes *undefined* if file consists of varying length records or is closed |
| **NEXTREC** | Integer*4 | Variable is assigned the value *n*+1 where *n* is the record number of the last record read or written on a file connected for direct access. If no record have been read or written the variable is set to 1. If the file is not connected for direct access or if the position of the file pointer is indeterminate due to a previous error the variable become undefined |
| **BLANK** | Character | **'ZERO'** non leading blanks in numeric fields will be converted to zeros<br>**'NULL** non leading blanks in numeric fields will be deleted<br>If the file is not open for formatted data transfer the variable becomes undefined |

▶ OPEN ([UNIT ]unit# [,FILE filename] [,STATUS stat] [,ACCESS acc] [,FORM fm] [,RECL reclngth] [,BLANK blnk] [,ERR label] [,IOSTAT ios])

An OPEN statement may be used to create a new file and establish its basic properties and/or to connect a file to a file unit and establish the properties of the connection.

The options used may be given in any order except that if **UNIT** is omitted **unit#** must appear first.

The OPEN statement options are defined as follows:

| Option | Argument Data Type | Results of Arguments Specified |
|---|---|---|
| UNIT | Integer*4 Expression | File is opened on the file unit specified. |
| FILE | Character Expression | The file has the name specified. A pathname may be used. If no FILE is specified for a new non scratch file, the file will be named F#nnn where nnn is the number of the file unit on which the file was opened. |
| STATUS | Character Expression | **OLD** Specified if the file already exists. |
| | | **NEW** Specified if the file is being created. |
| | | **SCRATCH** File is temporary; it will be automatically deleted at program end. No filename may be specified. |
| | | **UNKNOWN** (Default) Specified if the status is not known to the programmer. The processor will determine the appropriate status. |
| ACCESS | Character Expression | **SEQUENTIAL** (Default) File is connected for sequential access. |
| | | **DIRECT** (File is connected for direct access. |

| | | |
|---|---|---|
| **FORM=** | Character Expression | **'FORMATTED'** *(Default under sequential access)* File is connected for formatted data transfer |
| | | **'UNFORMATTED'**: (Default under direct access) File is connected for unformatted data transfer |
| **RECL=** | Integer*4 Expression | Sets record length for a file of fixed-length records. Must be omitted for a file of varying length records. Use in SAM files is an F77 extension. Required in DAM files. |
| **BLANK=** | Character Expression | This item specifies treatment of blanks in numeric input fields when data is read into the file. |
| | | **'NULL'**: *(Default)* All blanks are deleted and digits compressed to the right side of the input field. An all-blank field will be interpreted as a zero value. |
| | | **'ZERO'**: All but leading blanks are converted to zeroes as in FORTRAN 66. |
| **ERR=** | Statement Label | Control transfers to statement specified if an error occurs during execution of the OPEN statement. |
| **IOSTAT=** | Integer*4 Variable | Set to zero if the OPEN statement executes successfully. Set positive on error in OPEN statement execution. |

## ▶PRINT format [,output list]

A PRINT is a simplified WRITE equivalent to WRITE (1 format) [output list] The **format** is described under FORMATS below

## ▶READ Statement

| | |
|---|---|
| Sequential | READ ([UNIT ]unit# [ [FMT ]format ] [ END label] [ ERR label] [ IOSTAT ios]) [input list] |
| ANS direct | READ ([UNIT ]unit# [ [FMT ]format] REC record# [ END label] [ ERR label] [ IOSTAT ios]) [input list] |
| IBM direct | READ (unit # record# [ [FMT ]format] [ END label] [ ERR label] [ IOSTAT ios]) [input list] |

Transfers a record from the file open on **unit#** to the variables listed in **input list** The **format** is described under FORMATS below

The READ statement options are defined as follows

| Option | Argument Data Type | Results of Arguments Specified |
|---|---|---|
| UNIT | Integer Expression | Specifies the unit in which the file is open |
| FMT | See FORMATS below | If present the READ is formatted otherwise it is unformatted |
| END | Integer Constant | On endfile control will transfer to the indicated labelled statement |
| ERR | Integer Constant | On error control will transfer to the indicated labelled statement |
| IOSTAT | Integer Variable | The variable will be set to a positive value if an error occurs zero if the READ executes successfully and a negative value if end file was encountered and no error occurred |

## ▶REWIND ([UNIT- ]unit# [,IOSTAT ios] [,ERR= label])

The options are as described for the OPEN statement

Repositions the file pointer to the initial point of a file either by physically rewinding a tape, or by resetting a disc file's logical pointer

### ▶WRITE statement

| | |
|---|---|
| Sequential | WRITE ([UNIT ]unit# [ [FMT ]format ] [ ERR label] [ IOSTAT ios]) [output list] |
| ANS direct | WRITE ([UNIT ]unit# [ [FMT ]format ] REC record# [ ERR label] [ IOSTAT ios]) [output list] |
| IBM direct | WRITE (unit# record# [ [FMT ]format] [ ERR label] [ IOSTAT ios]) [output list] |

Transfers the values listed in **output list** to a record in the file open on **unit#**. The **format** is described under FORMATS below.

The WRITE statement options have the same meaning as the READ statement options. Since endfile is not possible there is no **END=** option and **IOSTAT=** will never receive a negative value.

# FORMATS

Formatted data transfer occurs when a **format** is given in a READ, WRITE or PRINT statement. The **format** designates a format list — a parenthesized list of I/O descriptors — which is to be used in formatting the data. A **format** may be:

- The statement number of a FORMAT statement.
- An INTEGER variable that has been ASSIGNed such a number.
- A fixed length (no adjustable-length components) character expression whose value is a format list.
- A CHARACTER array, array element, variable or constant whose value is a format list.
- An asterisk, denoting list directed I/O.

### Types of I/O descriptor

There are two types of I/O descriptors. Field descriptors which specify the conversions for individual data items and edit control descriptors which specify more general aspects of the data transfer process. Field descriptors are further subdivided into numeric and non-numeric descriptors.

### Numeric descriptors in general

The numeric field descriptors are D, E, F, G, and I. The following rules apply to all numeric descriptors:

1 Leading blanks are not significant for input For output leading zeroes are suppressed A minus sign is printed for a negative number but a positive number is left unsigned
2 For input with F E D and G descriptors a decimal point in the input field over-rides the *d* specification in the descriptor
3 For output fields are right justified If the field width is insufficient asterisks are produced
4 Excess digits of precision may be specified on input to non INTEGER numeric data types The excess will be ignored
5 See the BLANK- option of the OPEN statement for the rules concerning blanks in input fields

A complex number consists of a pair of real or double precision numbers It is edited with an appropriate pair of real or double precision field descriptors The fact that the two numbers form one entity mathematically is irrelevant to input/output Edit-control descriptors may appear between the two field descriptors

**Conventions**

The following conventions are used in the discussions of I O descriptors

| | |
|---|---|
| **w** | The size in characters of the external field to/from which the data is being transferred |
| **d** | The number of places to the right of the decimal point |
| **e** | The number of exponent digits to be displayed on output |
| **n** | Any integer in the range appropriate to the particular case |

**Descriptors**

**▶A[w]** **Character**

**w** is required for input but optional for output In the following **L** is the length of the character item being edited

| | |
|---|---|
| **Input:** | If **w** >= **L**, the rightmost L characters are taken from the external input field If **w** < **L**, the *w* characters are left justified in the data item and padded with blanks |
| **Output:** | If **w** > **L**, the characters are printed right justified in the field, preceded by blanks as needed If **w** <= **L**, the leftmost **w** characters are printed If **w** is not specified it is assumed to be equal to **L**. |

## ▶B 'string' Business

Prints templated numerical output for business purposes

**Features include:** Fixed and floating signs trailing signs, plus sign suppression, trailing minus change to CR' fixed and floating $, field filling leading zero suppression, and insertion of commas The length of the string determines the field width if the length is greater than the field width, the output is printed as a string of asterisks

**B Format Characters**

| String Symbol | Usage |
|---|---|
| + | Fixed Sign |
| + + | Floating sign |
| - | Fixed sign, plus sign suppression |
| - - | Floating sign, plus sign suppression |
| $ | Fixed currency sign |
| $ $ | Floating currency sign |
| Z | Print digits 1 9 replace leading zeroes with blanks |
| # | Print digits 0-9 |
| . | Position of decimal point |
| , | Position of comma |
| CR | Trailing blank (positive) or CR (negative) |
| * | Fill field with asterisks |

No repeat count is allowed on an individual B descriptor but a B descriptor may be included in a group that is repeated

### ▶BN BZ — Blank Control

The method of handling blanks in numeric input fields that is established for a file by the **BLANK** option of the OPEN statement may be temporarily overridden by **BN** or **BZ**. The method may be altered as often as desired and will revert to the **BLANK** value when the READ statement is complete. Blank control descriptors have no effect on output.

| | |
|---|---|
| **BN** | All blanks will be deleted and digits compressed to the right side of the input field. An all blank field is interpreted as a zero value. |
| **BZ** | All but leading blanks will be converted to zeroes, as in FORTRAN 66. |

### ▶Dw.d — Double Precision

Edits a double precision number.

| | |
|---|---|
| **Input** | Operates exactly like an E descriptor. |
| **Output** | Operates exactly like an E descriptor with no Ee present, except that a D is substituted wherever an E would appear in the output field. |

### ▶Ew.d[Ee] — Real (exponential)

Edits a REAL or DOUBLE PRECISION number with an exponent.

| | |
|---|---|
| **Input** | *The exponent may be omitted. E+00 will be assumed.* |
| **Output** | If **Ee** is present, e digits of the exponent will be printed.<br>If **Ee** is omitted, the appearance of the exponent will be as follows: |

| Value of Exponent | Appearance of Exponent |
|---|---|
| $-99 \leq exp \leq 99$ | E ± zz |
| $-999 \leq exp < -99$ | -zzz (no E) |
| $99 < exp \leq 999$ | +zzz (no E) |
| $-9999 < exp < -999$ | =zzz (fourth digit lost) |
| $999 < exp \leq 9999$ | $zzz (fourth digit lost) |

Note that the number is always normalized. For non-normalized output, use a scale factor.

## ▶Fw.d Real (non-exponential)

Writes a real number without an exponent. Reads any real or double precision number.

**w** is the size of the field including blanks, the sign and the decimal point.

**d** is the number of places to the right of the decimal point.

| | |
|---|---|
| **Input:** | The decimal point may be omitted from the field. The rightmost **d** digits will be interpreted as decimal digits. If a decimal point is present its position over-rides **d**. Input fields appropriate for E and D editing will also work for F editing. |
| **Output:** | **d** decimal positions are always written. |

## ▶Gw.d[Ee] Real (General)

Edits real data whose magnitude is too unpredictable to allow use of D, E or F.

| | |
|---|---|
| **Input:** | The G descriptor is equivalent to the F descriptor. |
| **Output:** | The G descriptor acts as follows: |

| Magnitude (M) of Real Data Item | G descriptor acts as. |
|---|---|
| 0.1 <= M < 1 | F(w-n) d nX |
| 1 <= M < 10 | F(w-n) (d-1) nX |
| 10 <= M < 100 | F(w-n) (d-2) nX |
| . . . | . . . |
| 10**(d-2) <= M < 10**(d-1) | F(w-n) (-1) nX |
| 10**(d-1) <= M < 10**(d) | F(w-n) 0 nX |
| Otherwise | E w.d[Ee] |

**where n** is 4 for Gw.d and e+2 for Gw.dEe

If M < 0.1 or M >= 10**d then Gw.d is equivalent to kPEw.d where **k** is the current scale factor.

For input the Gw.dEe field descriptor is treated identically to the Gw.d descriptor. For output Gw.dEe acts as Ew.dEe if 0.1 <= M < 10**d, and acts as Ew.d Ee otherwise.

**▶Iw[ n]** **Integer**

Edits a long or short integer. The **n** is the minimum number of places to be displayed on output. Leading zeroes will be printed if necessary.

**▶Lw** **Logical**

Edits a LOGICAL data item.

| | |
|---|---|
| **Input:** | A valid input field consists of optional blanks optionally followed by a decimal point, followed by a T or an F. The T or F may be followed by additional characters in the field; they will be ignored. |
| **Output:** | The output field consists of **w**-1 blanks followed by a T or F, as the value of the internal datum is true or false, respectively. |

**▶nP** **Scale Factor**

The scale factor **n** is an unsigned or negative integer constant. The comma following a P descriptor is often omitted so that it becomes a prefix of a subsequent field descriptor. The scale factor has various effects, depending on the descriptor type and the direction of data transfer.

**F, E, D, and G input** If there is an exponent in the field, the scale factor has no effect. Otherwise it converts the data so that

**External Value = Internal Value*(10**k)**

**F output** The scale factor converts the value as for F input.

**E and D output** The mantissa is multiplied by 10**k and the exponent is reduced by **k** to maintain the same overall value. This permits output of E and D numbers in non-normalized form.

**G output** If the G is acting as an F, the scale factor is ignored. If it is acting as an E, the scale factor behaves as described for E output.

Once a scale factor has been used, it remains in effect for all subsequent descriptors of appropriate type, until it is reset to another value or to zero. When a format list is rescanned, the scale factor is *not* reset to zero automatically. If a scale factor is to affect only one field, "0P" must appear before the next scalable descriptor that occurs.

### ▶SP SS S — Sign Control

These control the placement of plus signs in numeric output. Once a sign control descriptor is encountered it remains in effect until it is explicitly altered or revoked.

| | |
|---|---|
| **SP** | The processor will print a plus sign wherever one may optionally appear. |
| **SS** | The processor will not print any plus sign whose appearance is optional. |
| **S** | The processor will return to the locally defined system default for sign editing. |

### ▶Tn TLn TRn — Tab Control

These move the logical pointer which designates the next position in the record that will be read or written.

| | |
|---|---|
| **Tn** | Tab to column **n** of the record. |
| **TLn** | Tab **n** columns left of the current position. |
| **TRn** | Tab **n** columns right of the current position. |

If an attempt is made to tab off either end of the record the pointer will remain at the position adjacent to the end. Positions left undefined through use of the X descriptor for output will be filled with blanks.

### ▶nX — Space Skipping

On input equivalent to **TRn**. On output equivalent to a character constant of **n** blanks.

### ▶ : (Colon) — Conditional Output

A colon placed in a format list will cause data transfer to terminate at that point *if* no items remain in the output list. A colon is ignored on input.

### ▶ / (Slash) — Record-Skipping

A slash in a format list causes I/O processing to proceed to the next record. As many new records will be begun as there are slashes. The effect of slashes at the beginning or end of a format list is additional to the automatic beginning of a new record with each data transfer statement.

| | |
|---|---|
| **Input** | Under sequential access a slash causes the remaining portion of the current record to be skipped and the file pointer to be positioned at the beginning of the next record making it the current record Under direct access the remainder of the record is skipped the record number increased by one and the file pointer positioned at the beginning of the record that has that record number |
| **Output** | Similar to input except that all positions skipped over will be filled with blanks |

Commas adjacent to slashes may be omitted

**'ccc c** **Character Constant**

Each **c** is any ASCII character (not necessarily a member of the F77 character set)

A character string may appear as a constant in an output format list Such a string contains its own data obviating the need for a corresponding item in the output data list When the string is encountered during the scan of the format list the characters it contains are written to the current record A character constant may not appear in a format list used for input and may not be modified by an individual repeat count

**Carriage control**

The first character of each record in a file to be printed controls vertical spacing and is not printed The remaining characters in the record are printed starting at the left hand margin The significance of the permissable carriage control characters is

| Character | Vertical Spacing Before Printing |
|---|---|
| Blank | One line |
| 0 (zero) | Two lines |
| 1 | To first line of next page |
| + | No advance (overprint of last line) |

Records that contain no characters generated by slash editing or by an empty output list cause a blank line to be printed

### Repeat counts

A repeat count is an integer constant prefixed to a field descriptor, or to a parenthesized portion or the entirety of a format list. Individual edit-control descriptors can not have repeat counts. As data transfer proceeds, the format list items modified by the repeat count will be re-used the number of times specified before format control proceeds to subsequent format list items. Repeat counts have a maximum nesting of ten levels.

### Rescanning format lists

If the format list is exhausted before the I/O list, the file pointer is positioned at the beginning of the next record; format control then reverts to the beginning of the portion of the format list that was terminated by the last preceding right parenthesis. If there is no such parenthesis, format control reverts to the beginning of the format list. Any repeat count preceding the rescanned format is re-used. On output, the current record is padded with blanks and a new record started. On input, the remainder of the current record is skipped, and the file pointer advanced to the beginning of the next record. Reversion of format control, of itself, has no effect on the scale factor, the sign control (S, SP, SS), or the blank control (BN, BZ) in effect at the time of reversion.

# LIST-DIRECTED I/O

List directed I O occurs when an asterisk appears as the *format* in a READ WRITE or PRINT statement List-directed I O *cannot* be used in accessing internal files or DAM files

Adjacent values in a data line for list directed input must be separated by one or more blanks a comma or a slash Consecutive blanks are equivalent to single blanks Blanks adjacent to a comma or slash are of no significance An end of record is treated as a blank

Two adjacent commas with no intervening characters except blanks will leave the corresponding item in the input list unchanged A slash terminates a READ, leaving any remaining items in the input list unchanged A list directed READ continues until a slash is encountered or all the items in the input list have been satisfied If there are not enough values to complete the READ, an error will occur unless the data is being read from the terminal, in which case the program will wait for the remaining values to be typed in

Repeat counts may modify data items under list-directed input

**r*c**

represents r consecutive occurrences of the input value c If c is omitted r null values are read in leaving the next r elements of the input list unchanged No blanks may appear between r, *, and c

# INTRINSIC FUNCTIONS

It is impossible to fully describe the F77 intrinsic functions in the space available here Therefore it is important that the following be used only as a reminder not as a source of primary information

To get more information about a known function find it in the list of functions by name then proceed to the list of functions by category

To identify the function for a particular task locate that task in the list of functions by category

Most functions take one argument For each function taking more than one, there is a note describing the arguments required Where a note applies to a generic function it applies to all specific functions under the generic

Only named specific functions can be passed as arguments to subprograms Intrinsic functions for type conversion, selection of a maximum or minimum value lexical comparison, logical operation, shifting, truncation of bits, and determination of a data items memory address cannot be passed as arguments

## F77 intrinsic functions listed by name

| Function | Category |
|---|---|
| **ABS** | Absolute Value |
| **ACOS** | Arccosine |
| **AIMAG** | Imag Part Extraction |
| **AINT** | Truncation to Whole No |
| **ALOG** | Logarithm (Natural) |
| **ALOG10** | Logarithm (Common) |
| **AMAX1 (10)** | Largest Value |
| **AMAX0 (10)** | Largest Value |
| **AMIN1 (10)** | Smallest Value |
| **AMIN0 (10)** | Smallest Value |
| **AMOD (11)** | Remainder |
| AND (1) | AND (Bitwise) |
| **ANINT** | Nearest Whole Number |
| **ASIN** | Arcsine |
| **ATAN** | Arctangent |
| **ATAN2 (2)** | Arctangent of Quotient |
| **CABS** | Absolute Value |
| CCOS | Cosine |
| **CDABS** | Absolute Value |
| CDCOS | Cosine |
| **CDEXP** | Exponentiation |
| CDLOG | Logarithm (Natural) |
| **CDSQRT** | Square Root |
| **CDSIN** | Sine |
| **CEXP** | Exponentiation |
| CHAR | Conversion to Character |
| **CLOG** | Logarithm (Natural) |
| **CMPLX (3)** | Conversion to Complex |
| **CONJG** | Conjugate |
| COS | Cosine |
| **COSH** | Hyperbolic Cosine |
| CSIN | Sine |
| **CSQRT** | Square Root |
| DABS | Absolute Value |
| **DACOS** | Arccosine |
| DASIN | Arcsine |
| **DATAN** | Arctangent |
| **DATAN2 (2)** | Arctangent of Quotient |

| | |
|---|---|
| **DBLE** | Conversion to Dble Prec |
| **DCMPLX (3)** | Conversion to Complex *16 |
| **DCONJG** | Conjugate |
| **DCOS** | Cosine |
| **DCOSH** | Hyperbolic Cosine |
| **DDIM (4)** | Positive Difference |
| **DEXP** | Exponentiation |
| **DIM (4)** | Positive Difference |
| **DIMAG** | Imag Part Extraction |
| **DINT** | Truncation to Whole No |
| **DLOG** | Logarithm (Natural) |
| **DLOG10** | Logarithm (Common) |
| **DMAX1 (10)** | Largest Value |
| **DMINI (10)** | Smallest Value |
| **DMOD (11)** | Remainder |
| **DNINT** | Nearest Whole Number |
| **DPROD (5)** | Product (Double Precision) |
| **DREAL** | Conversion to Dble Prec |
| **DREAL** | Real Part Extraction |
| **DSIGN (13)** | Sign Transfer |
| **DSIN** | Sine |
| **DSINH** | Hyperbolic Sine |
| **DSQRT** | Square Root |
| **DTAN** | Tangent |
| **DTANH** | Hyperbolic Tangent |
| **EXP** | Exponentiation |
| **FLOAT** | Conversion to Real |
| **IABS** | Absolute Value |
| **ICHAR** | Conversion to Integer |
| **IDINT** | Conversion to Integer |
| **IDIM (4)** | Positive Difference |
| **IDNINT** | Nearest Integer |
| **IFIX** | Conversion to Integer |
| **INDEX (6)** | Index of a Substring |
| **INT** | Conversion to Integer |
| **INTS** | Conversion to Short Integer |
| **INTL** | Conversion to Long Integer |
| **ISIGN (13)** | Sign Transfer |
| **LEN** | Length of String |
| **LGE (7)** | Lexically >= |

| | |
|---|---|
| **LGT (7)** | Lexically > |
| **LLE (7)** | Lexically <= |
| **LLT (7)** | Lexically < |
| **LOC** | Location in Memory |
| **LOG** | Logarithm (Natural) |
| **LOG10** | Logarith (Common) |
| **LS (8)** | Shift Left |
| **LT (9)** | Truncate Left |
| **MAX (10)** | Largest Value |
| **MAX0 (10)** | Largest Value |
| **MAX1 (10)** | Largest Value |
| **MIN (10)** | Smallest Value |
| **MIN0 (10)** | Smallest Value |
| **MIN1 (10)** | Smallest Value |
| **MOD (11)** | Remainder |
| **NINT** | Nearest Integer |
| **NOT** | NOT (Bitwise) |
| **OR (1)** | OR (Bitwise) |
| **REAL** | Conversion to Real |
| **REAL** | Real Part Extraction |
| **RS (8)** | Shift Right |
| **RT (9)** | Truncate Right |
| **SHFT (12)** | Shift |
| **SIGN (13)** | Sign Transfer |
| **SIN** | Sine |
| **SINH** | Hyperbolic Sine |
| **SNGL** | Conversion to Real |
| **SQRT** | Square Root |
| **TAN** | Tangent |
| **TANH** | Hyperbolic Tangent |
| **XOR (1)** | XOR (Bitwise) |

## F77 intrinsic functions listed by category

| Category | Generic | Specific | Arg. type | Result type |
|---|---|---|---|---|
| **Absolute Value** | **ABS** | IABS | Integer | Integer |
| | | ABS | Real | Real |
| | | DABS | Double | Double |
| | | CABS | Complex | Real |
| | | CDABS | Complex*16 | Double |
| **Arccosine** | **ACOS** | ACOS | Real | Real |
| | | DACOS | Double | Double |
| **Arcsine** | **ASIN** | ASIN | Real | Real |
| | | DASIN | Double | Double |
| **Arctangent** | **ATAN** | ATAN | Real | Real |
| | | DATAN | Double | Double |
| **Arctangent of Quotient** | **ATAN2** (2) | ATAN2 | Real | Real |
| | | DATAN2 | Double | Double |
| **AND (bit)** | — | AND (1) | Integer | Integer |
| **Conjugate** | **CONJG** | CONJG | Complex | Complex |
| | | DCONJG | Complex*16 | Complex*16 |
| **Conversion to Character** | — | CHAR | Integer | Character |
| **Conversion to Complex** | **CMPLX** (3) | — | Integer | Complex |
| | | — | Real | Complex |
| | | — | Double | Complex |
| | | — | Complex | Complex |
| | | — | Complex*16 | Complex |
| **Conversion to Complex*16** | **DCMPLX** (3) | — | Integer | Complex*16 |
| | | — | Real | Complex*16 |
| | | — | Double | Complex*16 |
| | | — | Complex | Complex*16 |
| | | — | Complex*16 | Complex*16 |
| **Conversion to Double Precision** | **DBLE** | — | Integer | Double |
| | | — | Real | Double |
| | | — | Double | Double |
| | | — | Complex | Double |
| | | DREAL | Complex*16 | Double |
| **Conversion to Integer** | **INT** | — | Integer | Integer |
| | | INT | Real | Integer |
| | | IFIX | Real | Integer |
| | | IDINT | Double | Integer |
| | | — | Complex | Integer |
| | | — | Complex*16 | Integer |
| **Conversion to Integer** | — | ICHAR | Character | Integer |
| **Conversion to Long Integer** | **INTL** | — | Integer | Integer*4 |
| | | — | Real | Integer*4 |
| | | — | Double | Integer*4 |
| | | — | Complex | Integer*4 |
| | | — | Complex*16 | Integer*4 |

| | | | | |
|---|---|---|---|---|
| Conversion to Real | REAL | FLOAT | Integer | Real |
| | | — | Real | Real |
| | | SNGL | Double | Real |
| | | — | Complex | Real |
| | | REAL | Complex*16 | Real |
| Conversion to Short Integer | INTS | — | Integer | Integer*2 |
| | | — | Real | Integer*2 |
| | | — | Double | Integer*2 |
| | | — | Complex | Integer*2 |
| | | — | Complex*16 | Integer*2 |
| Cosine | COS | COS | Real | Real |
| | | DCOS | Double | Double |
| | | CCOS | Complex | Complex |
| | | CDCOS | Complex*16 | Complex*16 |
| Exponentiation | EXP | EXP | Real | Real |
| | | DEXP | Double | Double |
| | | CEXP | Complex | Complex |
| | | CDEXP | Complex*16 | Complex*16 |
| Hyperbolic Cosine | COSH | COSH | Real | Real |
| | | DCOSH | Double | Double |
| Hyperbolic Sine | SINH | SINH | Real | Real |
| | | DSINH | Double | Double |
| Hyperbolic Tangent | TANH | TANH | Real | Real |
| | | DTANH | Double | Double |
| Imag. Part Extraction | — | AIMAG | Complex | Real |
| | | DIMAG | Complex*16 | Double |
| Index of a Substring | — | INDEX (6) | Character | Integer |
| Largest Value | MAX (10) | MAX0 | Integer | Integer |
| | | AMAX1 | Real | Real |
| | | DMAX1 | Double | Double |
| Largest Value | — | AMAX0 (10) | Integer | Real |
| | | MAX1 (10) | Real | Integer |
| Length | — | LEN | Character | Integer |
| Lexically >= | — | LGE (7) | Character | Logical |
| Lexically > | — | LGT (7) | Character | Logical |
| Lexically <= | — | LLE (7) | Character | Logical |
| Lexically < | — | LLT (7) | Character | Logical |
| Location in Memory | — | LOC | Any but CHAR or LOG*1 | Integer*4 |
| Logarithm (common) | LOG10 | ALOG10 | Real | Real |
| | | DLOG10 | Double | Double |

| | | | | |
|---|---|---|---|---|
| **Logarithm (natural)** | **LOG** | ALOG | Real | Real |
| | | DLOG | Double | Double |
| | | CLOG | Complex | Complex |
| | | CDLOG | Complex*16 | Complex*16 |
| **Nearest Integer** | **NINT** | NINT | Real | Integer |
| | | IDNINT | Double | Integer |
| **Nearest Whole No.** | **ANINT** | ANINT | Real | Real |
| | | DNINT | Double | Double |
| **NOT (bit)** | — | NOT | Integer | Integer |
| **OR (bit)** | — | OR (1) | Integer | Integer |
| **Positive Difference** | **DIM** (4) | IDIM | Integer | Integer |
| | | DIM | Real | Real |
| | | DDIM | Double | Double |
| **Product (Doub. Prec.)** | — | DPROD (5) | Real | Double |
| **Real Part Extraction** | — | REAL | Complex | Real |
| | | DREAL | Complex*16 | Double |
| **Remainder** | **MOD**(11) | MOD | Integer | Integer |
| | | AMOD | Real | Real |
| | | DMOD | Double | Double |
| **Shift** | — | SHFT (12) | Integer | Integer |
| **Shift Left** | — | LS (8) | Integer | Integer |
| **Shift Right** | — | RS (8) | Integer | Integer |
| **Sign Transfer** | **SIGN** (13) | ISIGN | Integer | Integer |
| | | SIGN | Real | Real |
| | | DSIGN | Double | Double |
| **Sine** | **SIN** | SIN | Real | Real |
| | | DSIN | Double | Double |
| | | CSIN | Complex | Complex |
| | | CDSIN | Complex*16 | Complex*16 |
| **Smallest Value** | **MIN**(10) | MIN0 | Integer | Integer |
| | | AMIN1 | Real | Real |
| | | DMIN1 | Double | Double |
| **Smallest Value** | — | AMIN0 (10) | Integer | Real |
| | | MIN1 (10) | Real | Integer |
| **Square Root** | **SQRT** | SQRT | Real | Real |
| | | DSQRT | Double | Double |
| | | CSQRT | Complex | Complex |
| | | CDSQRT | Complex*16 | Complex*16 |
| **Tangent** | **TAN** | TAN | Real | Real |
| | | DTAN | Double | Double |
| **Truncation to Whole No.** | **AINT** | AINT | Real | Real |
| | | DINT | Double | Double |
| **Truncate Left** | — | LT (9) | Integer | Integer |
| **Truncate Right** | — | RT (9) | Integer | Integer |
| **XOR (bit)** | — | XOR (1) | Integer | Integer |

**Notes for multi-argument functions**

1. Any number of arguments.
2. Two arguments. Returns the arctangent (in radians) of their quotient.
3. One or two arguments. With one argument, the argument becomes the real part, and the imaginary part is zero. With two arguments, ARG1 becomes the real part, and ARG2 the imaginary part.
4. Two arguments. ARG2 is subtracted from ARG1. If the difference is positive, it is returned; if not, the value zero is returned.
5. Two arguments. The arguments are multiplied and the result is returned in DOUBLE PRECISION form.
6. Two CHARACTER arguments. If ARG2 is a substring of ARG1, the position in ARG1 where ARG2 begins is returned. If not, the value zero is returned.
7. Two CHARACTER arguments. If they have the specified relationship in the ASCII collating sequence, .TRUE. is returned; otherwise .FALSE. is returned.
8. Two arguments. Shifts ARG1 by the number of bits specified in ARG2. Vacated places are filled with zeroes.
9. Two arguments. Preserves the left (LT) or right (RT) ARG2 bits of ARG1, and sets the rest to zero.
10. Takes any number of arguments.
11. Two arguments. Returns the remainder when ARG1 is divided by ARG2.
12. Two or three arguments. Similar to LS and RS (see Note 8) except that it can shift in either direction and can perform two shifts rather than one.
    If ARG2 is negative, the first shift is to the left; if it is positive, the shift is to the right; if it is zero, no

shift occurs
If ARG3 appears, the shift specified by it will occur after the shift specified by ARG2 is complete

13 Two arguments The value returned has the magnitude of ARG1 and the sign of ARG2 If ARG1 is zero the result is zero, which is neither positive or negative

# THE F77 COMPILER

The F77 compiler is invoked by the command

**F77 pathname [-option] .**

| | |
|---|---|
| **pathname** | The pathname of the FORTRAN 77 source program to be compiled |
| **options** | Mnemonics for the options controlling compiler functions |

The F77 compiler options are as follows In each case, the abbreviation for each option is in rust and the default is underlined

**-BIG / -NOBIG**

Determines code generated for dummy array references in a subprogram

**-B[INARY] [argument]**

The argument may be

| | |
|---|---|
| **pathame** | Object code will be written to the file **pathname** |
| **YES** | Object code will be written to the file named B_ program, where **program** is the name of the source file |
| **NO** | No binary file will be created Specified when only a syntax check is desired |

When no -B option is given, or -B without an argument is given -B YES will be presumed

**-DCLVAR / -NODCLVAR**

Controls flagging of undeclared variables

***-DEBUG / -NODEBUG***

Controls generation of code allowing the program to run under the symbolic debugger

**-DO1 / -NOD01**

Controls the type of DO-loop which the compiler will produce

**-DYNM / -SAVE**

Determines data storage mode dynamic or static

**-ERRLIST / -NOERRLIST**

Controls generation of an errors-only file The file will be named as described under -L YES

**-ERRTTY / -NOERRTTY**

Controls printing of error messages at the terminal

**-EXPLIST / -NOEXPLIST (Implies -L)**

Controls insertion of a pseudo-assembly code listing into the source listing

**-INTI / -INTS**

Determines default lengths for type INTEGER data items whose length is not explicitly declared

**-L[ISTING] [argument]**

Controls creation of the source listing file The argument may be

| | |
|---|---|
| **pathname** | Listing will be written to the file *pathname* |
| **YES** | Listing will be written to a file named L_program, where *program* is the name of the source file |
| **TTY** | The listing will be printed at the user terminal |
| **SPOOL** | The listing will be spooled directly to the line printer Default SPOOL arguments are in effect |
| **NO** | No listing file will be generated |

When no -L option is given -I NO will be presumed When -L is given with no argument -L YES will be presumed

**-LOGL / -LOGS**

Determines default lengths for type LOGICAL data items whose length is not explicitly declared, and for the logical constants

**-OPTIMIZE / -NOOPTIMIZE**

Controls the optimization phase of the compiler

**-PRODUCTION / -NOPRODUCTION**

Alternative option controlling code for the debugger. -PRODUCTION is similar to -DEBUG, except that the code generated will not permit insertion of statement break points

**-RANGE / -NORANGE**

Controls error checking for out-of-bounds values of array subscripts and character substring indexes

**-SILENT / -NOSILENT**

Suppresses WARNING messsages

**-STATISTICS / -NOSTATISTICS**

Controls printout of compiler statistics

**-UPCASE / -LCASE**

Controls mapping of lowercase to uppercase letters in a source program

**-XREF / -NOXREF (Implies -L)**

Controls generation of a cross reference

**-64V / -32I**

Controls addressing mode to be used in the object code

# ASCII COLLATING SEQUENCE

ASCII Character Set (Printing)

| Octal Value | ASCII Character | Octal Value | ASCII Character | Octal Value | ASCII Character |
|---|---|---|---|---|---|
| 240 | SP | 300 | @ | 340 | ` |
| 241 | ! | 301 | A | 341 | a |
| 242 | " | 302 | B | 342 | b |
| 243 | # | 303 | C | 343 | c |
| 244 | $ | 304 | D | 344 | d |
| 245 | % | 305 | E | 345 | e |
| 246 | & | 306 | F | 346 | f |
| 247 | ' | 307 | G | 347 | g |
| 250 | ( | 310 | H | 350 | h |
| 251 | ) | 311 | I | 351 | i |
| 252 | * | 312 | J | 352 | j |
| 253 | + | 313 | K | 353 | k |
| 254 | , | 314 | L | 354 | l |
| 255 | - | 315 | M | 355 | m |
| 256 | . | 316 | N | 356 | n |
| 257 | / | 317 | O | 357 | o |
| 260 | 0 | 320 | P | 360 | p |
| 261 | 1 | 321 | Q | 361 | q |
| 262 | 2 | 322 | R | 362 | r |
| 263 | 3 | 323 | S | 363 | s |
| 264 | 4 | 324 | T | 364 | t |
| 265 | 5 | 325 | U | 365 | u |
| 266 | 6 | 326 | V | 366 | v |
| 267 | 7 | 327 | W | 367 | w |
| 270 | 8 | 330 | X | 370 | x |
| 271 | 9 | 331 | Y | 371 | y |
| 272 | : | 332 | Z | 372 | z |
| 273 | ; | 333 | [ | 373 | { |
| 274 | < | 334 | \ | 374 | \| |
| 275 | = | 335 | ] | 375 | } |
| 276 | > | 336 | ^ | 376 | ~ |
| 277 | ? | 337 | _ | 377 | DEL |

# POWERS OF TWO

| 2 | n | 5 |
|---|---|---|
| 1 | 0 | 1 |
| 2 | 1 | 5 |
| 4 | 2 | 25 |
| 8 | 3 | 125 |
| 16 | 4 | 0625 |
| 32 | 5 | 03125 |
| 64 | 6 | 015625 |
| 128 | 7 | 0078125 |
| 256 | 8 | 00390625 |
| 512 | 9 | 001953125 |
| 1024 | 10 | 0009765625 |
| 2048 | 11 | 00048828125 |
| 4096 | 12 | 000244140625 |
| 8192 | 13 | 0001220703125 |
| 16384 | 14 | 00006103515625 |
| 32768 | 15 | 000030517578125 |
| 65536 | 16 | 0000152587890625 |
| 131072 | 17 | 0000076293945312 5 |
| 262144 | 18 | 00000381469726562 5 |
| 524288 | 19 | 000001907348632812 5 |
| 1048576 | 20 | 0000009536743164062 5 |
| 2097152 | 21 | 0000004768371582031 25 |
| 4194304 | 22 | 0000002384185791015625 |
| 8388608 | 23 | 0000001192092895507812 5 |
| 16777216 | 24 | 0000000596046447753906 25 |
| 33554432 | 25 | 0000000298023223876953 125 |
| 67108864 | 26 | 0000000149011611938476 5125 |
| 134217728 | 27 | 0000000074505805969238 25625 |
| 268435456 | 28 | 0000000037252902984619 128125 |
| 536870912 | 29 | 0000000018626451492 3095640625 |
| 1073741824 | 30 | 0000000009313225746154 78203125 |
| 2147483648 | 31 | 0000000004656612873077 391015625 |
| 4294967296 | 32 | 0000000002328306518653 86955078125 |

Prime Computer, Inc., Technical Publications Department
500 Old Connecticut Path, Framingham, MA 01701

FDR4030-000