Chapter 1

Fortran Elements and Concepts

This chapter contains the following subsections:

- "Fortran Character Set"
- "Data Types"
- "Collating Sequence"
- "Symbolic Names"
- "Variables"
- "Source Program Lines"
- "Statements"
- "Program Units"
- "Program Organization"

This chapter provides definitions for the various elements of a Fortran program. The Fortran language is written using a specific set of characters that form the words, numbers, names, and expressions that make up Fortran statements. These statements form a Fortran program. The Fortran character set, the rules for writing Fortran statements, the main structural elements of a program, and the proper order of statements in a program are also discussed in this chapter.

Fortran Character Set

The Fortran character set consists of 26 uppercase and 26 lowercase letters (*alphabetic* characters), the numbers 0 through 9 (*digits*), and *special* characters. This manual refers to letters (uppercase and lowercase) together with the underscore (_) as *extended alphabetic* characters. The *extended alphabetic* characters together with the digits are also referred to as *alphanumeric* characters. The complete character set is

Letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits:	0 1 2 3 4 5 6 7 8 9

Special Characters:

	Blank
=	Equal
+	Plus
–	Minus
*	Asterisk

/	Slash
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Decimal point
$	Currency symbol
'	Apostrophe
:	Colon
!	Exclamation point
_	Underscore
"	Quotation mark

Lowercase alphabetic characters, the exclamation point (!), the underscore (_), and the double quote (") are extensions to Fortran 77. Digits are interpreted in base 10. A special character can serve as an operator, a part of a character constant, a part of a numeric constant, or some other function

Blank Characters

Use blank characters freely to improve the appearance and readability of Fortran statements. They have no significance in Fortran statements, except

- in character constants
- for H and character editing in format specifications
- in Hollerith constants
- to signify an initial line when used in column 6 of source line
- when counting the total number of characters allowed in any one statement

These special considerations are discussed in detail in later sections.

Escape Sequences

Table 1−1lists escape sequences for representing non−graphic characters and for compatibility with the C programming language.

Table 1−1 C Escape Sequences

Sequence	Meaning
\n	New line
\t	Tab
\b	Backspace
\f	Form feed

\0	Null
\'	Apostrophe (does not terminate a string)
\"	Quotation mark (does not terminate a string)
\\	\
\x	x represents any character

The compiler treats the backslash character as the beginning of an escape sequence by default. To use backslash as a normal character, compile the program with the **-backslash** option.

Data Types

In general, there are three kinds of entities that have a data type: constants, data names, and function names. The types of data allowed in Fortran are

- **INTEGER**—positive and negative integral numbers and zero
- **REAL**—positive and negative numbers with a fractional part and zero
- **DOUBLE PRECISION**—same as **REAL** but using twice the storage space and possibly greater precision
- **COMPLEX**—ordered pair of **REAL** data: real and imaginary components
- **DOUBLE COMPLEX**—ordered pair of double–precision data
- **LOGICAL**—Boolean data representing *true* or *false*
- **CHARACTER**—character strings
- **HOLLERITH**—an historical data type for character definition

Together, **INTEGER**, **REAL**, **DOUBLE PRECISION**, **COMPLEX**, and **DOUBLE COMPLEX** constitute the class of *arithmetic* data types.

The type of data is established in one of two ways: implicitly, depending on the first letter of its symbolic name (described in this chapter), or explicitly through a type statement (described in Chapter 4). A data value can be a variable or a constant, that is, its value either can or cannot change during the execution of a program. An array is a sequence of data items occupying a set of consecutive bytes.

If not explicitly specified by a type statement or a **FUNCTION** statement, the data type of a data item, data name, or function name is determined implicitly by the first character of its symbolic name. By default, symbolic names beginning with I, J, K, L, M, or N (uppercase or lowercase) imply an **INTEGER** data type; names beginning with all other letters imply a **REAL** data type. You can change or confirm the default implicit data type corresponding to each letter of the alphabet through an **IMPLICIT** statement (refer to "EXTERNAL" of Chapter 4 for details).

The data type of external functions and statement functions is implicitly determined in the same manner as above. The type of an external function can also be explicitly declared in a **FUNCTION** statement.

Collating Sequence

The Fortran collating sequence defines the relationship between letters and digits and is used when comparing character strings. The collating sequence is determined by these rules:

- A is less than Z, and a is less than z. The listing order of the alphabetic characters specifies the collating sequence for *alphabetic* characters. The relationship between lowercase and uppercase of the same letter is unspecified.
- 0 is less than 9. The order in which digits are listed above defines the collating sequence for digits.
- Alphabetic characters and digits are not intermixed in the collating sequence.
- The blank character is less than the letter A (uppercase and lowercase) and less than the digit 0.
- The special characters given as part of the character set are *not* listed in any specific order. There is no specification as to where special characters occur in the collating sequence.

Symbolic Names

A symbolic name is a sequence of characters that refer to a memory location by describing its contents. Symbolic names identify the following user–defined local and global entities:

Local	variableconstantarraystatement functionintrinsic functiondummy procedure
Global	common blockexternal functionsubroutinemain programblock data subprogram

Conventions

A symbolic name can contain any alphanumeric character; digits and _ (underscore) are allowed in addition to uppercase and lowercase alphabetic characters. However, the first character *must* be a letter.

- Fortran symbolic names can contain any number of characters, but only the first 32 of these are significant in distinguishing one symbolic name from another (standard Fortran 77 allows only 6 characters.) Symbolic names that are used externally (program names, subroutine names, function names, common block names) are limited to 32 significant characters.
- The inclusion of the special period (.), underscore (_), and dollar sign ($) characters in symbolic names is an enhancement to Fortran 77.

Examples of valid symbolic names are

```
CASH  C3P0 R2D2  LONG_NAME
```

Examples of invalid symbolic names are

```
X*4     (contains a special character, *)
3CASH   (first character is a digit)
```

Data Types of Symbolic Names

A symbolic name has a definite data type in a program unit that can be any of the following:

- **BYTE**

- **INTEGER [*1 | *2 | *4]**
- **REAL [*4 | *8**] or **DOUBLE PRECISION**
- **COMPLEX [*8 | *16]**
- **LOGICAL [*1 | *2 | *4]**
- **CHARACTER [***n**]**

The optional length specifier that follows the type name determines the number of bytes of storage for the data type. If the length specifier is omitted, the compiler uses the defaults listed in the *Fortran 77 Programmer's Guide*.

In general, wherever the usage of a given data type is allowed, it can have any internal length. One exception is the use of integer variables for assigned **GOTO** statements. In this case the integer variable must be 4 bytes in length.

Data of a given type and different internal lengths can be intermixed in expressions, and the resultant value will be the larger of the internal representations in the expression.

Note: The lengths of arguments in actual and formal parameter lists and **COMMON** blocks must agree in order produce predictable results.

Scope of Symbolic Names

The following rules determine the scope of symbolic names:

- A symbolic name that identifies a global entity, such as a common block, external function, subroutine, main program, or block data subprogram, has the scope of an executable program. Do not use it to identify another global entity in the same executable program.
- A symbolic name that identifies a local entity, such as an array, variable, constant, statement function, intrinsic function, or dummy procedure, has the scope of a single program unit. Do not use it to identify any other local entity in the same program unit.
- Do not use a symbolic name assigned to a global entity in a program unit for a local entity in the same unit. However, you can use the name for a common block name or an external function name that appears in a **FUNCTION** or **ENTRY** statement.

Variables

A variable is an entity with a name, data type, and value. Its value is either defined or undefined at any given time during program execution.

The variable name is a symbolic name of the data item and must conform to the rules given for symbolic names. The type of a variable is explicitly defined in a type−statement or implicitly by the first character of the name.

A variable cannot be used or referred to unless it has been defined through an assignment statement, input statement, **DATA** statement, or through association with a variable or array element that has been defined.

Source Program Lines

A source program line is a sequence of character positions, called *columns*, numbered consecutively starting from column 1 on the left.

Format

The two formats for Fortran programs are

- Fixed format—based on columns
- TAB format—based on the tab character

Fixed Format

A Fortran line is divided into columns, with one character per column as indicated in Table 1–2

Table 1–2 Fortran Line Structure

Field	Column
Statement label	1 through 5
Continuation indicator	6
Statement	7 to the end of the line or to the start of the comment field
Comment (optional)	73 through end of line

The **-col72**, **-col120**, **-extend_source**, and **-noextend_source** command line options are provided to change this format. See Chapter 1 of the *Fortran77 Programmer's Guide* for details. Several of these options can be specified in–line as described in Chapter 11, "Compiler Options."

TAB Character Formatting

Rather than aligning characters in specific columns, the TAB character can be used as an alternative field delimiter, as follows:

1. Type the statement label and follow it with a TAB. If there is no statement label, start the line with a TAB.
2. After the TAB, type either a statement initial line or a continuation line. A continuation line must contain a digit (1 through 9) immediately following the TAB. If any character *other* than a nonzero digit follows the TAB, the line will be interpreted as an initial line.
3. In a continuation line beginning with a TAB followed by a nonzero digit, any characters following the digit to the end of the line are a continuation of the current statement.
4. TAB–formatted lines do not have preassigned comment fields. All characters to the end of the line are considered part of the statement. However, you can use an exclamation point (!) to explicitly define a comment field. The comment field extends from the exclamation point to the end of the line.

The rules for TAB formatting can be summarized as *statement label* TAB *statement* (initial line)

TAB *continuation field statement* (continuation line)

TAB *statement* (initial line)

Note that although many terminal and text editors advance the cursor to a predetermined position when a TAB is entered, this action is not related to how the TAB will be ultimately interpreted by the compiler. The compiler interprets TABs in the statement field as blanks.

Types of Lines

The four types of Fortran program lines are

- comment
- debugging (an extension to Fortran 77)
- initial
- continuation

Comments

A comment line is used solely for documentation purposes and does not affect program execution. A comment line can appear anywhere in a program and has one of the following characteristics:

- An uppercase **C** or an asterisk (*) in column 1 and any sequence of characters from column 2 through to the end of the line
- A blank line
- Text preceded by an exclamation point (!) at any position of the line

Debugging Lines

Specify a **D** in column 1 for debugging purposes; it conditionally compiles source lines in conjunction with the **-d_lines** option described in Chapter 1 of the *Fortran 77 Programmer's Guide*. When the option is specified at compilation, all lines with a **D** in column 1 are treated as lines of source code and compiled; when the option is omitted, all lines with a **D** in column 1 are treated as comments.

Initial Lines

Initial lines contain the Fortran language statements that make up the source program; these statements are described in detail in "Program Organization". These fields divide each Fortran line into

- statement label field
- continuation indicator field
- statement field
- comment field

The fields in a Fortran line can be entered either on a character−per−column basis or by using the TAB character to delineate the fields, as described in the previous section.

Continuation Lines

A continuation line continues a Fortran statement and is identified as follows:

- Columns 1 through 5 must be blank.
- Column 6 contains any Fortran character other than a blank or the digit 0. Column 6 is frequently used to number the continuation lines.

As with initial lines, columns 7 through the end of the line contain the Fortran statement or a continuation of the statement.

Alternatively, you can use an ampersand (**&**) in column 1 to identify a continuation line. Using an **&** in column 1 implies that columns 2 through the end of the line are part of the statement. In Fortran 77, any remaining columns (column 73 and on) of a continuation line are not interpreted.

The maximum number of consecutive continuation lines is 99 unless you change this limit with the **-NC** compiler option.

Statements

Fortran statements are used to form program units. All Fortran statements, except assignment and statement functions, begin with a keyword. A *keyword* is a sequence of characters that identifies the type of Fortran statement.

A statement cannot begin on a line that contains any portion of a previous statement, except as part of a logical **IF** statement.

The **END** statement signals the physical end of a Fortran program unit and begins in column 7 or any later column of an initial line.

Statement Labels

A statement label allows you to refer to individual Fortran statements. A statement label consists of one to five digits—one of which must be nonzero—placed anywhere in columns 1 through 5 of an initial line. Blanks and leading zeros are not significant in distinguishing between statement labels.

The following statement labels are equivalent:

```
" 123 "   "123  "   "1 2 3"   "00123"
```

Each statement label must be unique within a program unit.

Fortran statements do not require labels. However, only labeled statements can be referenced by other Fortran statements. Do not label **PROGRAM**, **SUBROUTINE**, **FUNCTION**, **BLOCK DATA**, or **INCLUDE** statements.

Executable Statements

An executable statement specifies an identifiable action and is part of the execution sequence, (described in "Program Organization") in an executable program.

The three classes of executable statements are

- assignment statements
  - arithmetic
  - logical
  - statement label (**ASSIGN**)
  - character assignment
- control statements
  - unconditional, assigned, and computed **GO TO**
  - arithmetic **IF** and logical **IF**
  - block **IF**, **ELSE IF**, **ELSE**, and **END IF**
  - **CONTINUE**
  - **STOP** and **PAUSE**
  - **DO**
  - **CALL** and **RETURN**
  - **END**
- I/O statements
  - **READ**, **WRITE**, and **PRINT**
  - **REWIND**, **BACKSPACE**, **ENDFILE**, **OPEN**, **CLOSE**, and **INQUIRE**
  - **ACCEPT**, **TYPE**, **ENCODE**, **DECODE**, **DEFINE FILE**, **FIND**, **REWRITE DELETE**, and **UNLOCK**

Non−executable Statements

A non−executable statement is not part of the execution sequence. You can specify a statement label on most types of non−executable statements, but you cannot also specify that label for an executable statement in the same program unit.

A non−executable statement can perform one of these functions:

- Specify the characteristics, storage arrangement, and initial values of data
- Define statement functions
- Specify entry points within subprograms
- Contain editing or formatting information

- Classify program units
- Specify inclusion of additional statements from another source

The following data type statements are classified as non−executable:

- **CHARACTER**
- **COMPLEX**
- **DIMENSION**
- **DOUBLE PRECISION**
- **INTEGER**
- **LOGICAL**
- **REAL**
- **BYTE**

Additional non−executable program statements are

BLOCK DATA	**INCLUDE**
COMMON	**INTRINSIC**
DATA	**PARAMETER**
ENTRY	**POINTER**
EQUIVALENCE	**PROGRAM**
EXTERNAL	**SAVE**
FORMAT	**SUBROUTINE**
FUNCTION	Statement function
IMPLICIT	**VIRTUAL**

Program Units

Fortran programs consist of one or more program units. A program unit consists of a sequence of statements and optional comment lines. It can be a main program or a subprogram. The program unit defines the scope for symbolic names and statement labels.

The **END** statement must always be the last statement of a program unit.

Main Program

The main program is the program unit that initially receives control on execution. It can have a **PROGRAM** statement as its first statement and contain any Fortran statement except a **FUNCTION**, **SUBROUTINE**, **BLOCK DATA**, **ENTRY**, or **RETURN** statement. A **SAVE** statement in a main program does not affect the status of variables or arrays. A **STOP** or **END** statement in a main program terminates execution of the program.

The main program does not need to be a Fortran program. Refer to the *Fortran 77 Programmer's Guide* for information on writing Fortran programs that interact with programs written in other languages.

The main program cannot be referenced from a subprogram or from itself.

Subprograms

A subprogram is an independent section of code designed for a specialized purpose. It receives control when referenced or called by a statement in the main program or another subprogram.

A subprogram can be a

- function subprogram identified by a **FUNCTION** statement
- subroutine subprogram identified by a **SUBROUTINE** statement
- block data subprogram identified by a **BLOCK DATA** statement
- non−Fortran subprogram

Subroutines, external functions, statement functions, and intrinsic functions are collectively called procedures. A *procedure* is a program unit that performs an operational function.

An external procedure is a function or subroutine subprogram that is processed independently of the calling or referencing program unit. It can be written as a non−Fortran subprogram as described in the *Fortran 77 Programmer's Guide.*

Intrinsic functions are supplied by the processor and are generated as in−line functions or library functions. Refer to Appendix A, "Intrinsic Functions," for a description of the functions, the results given by each, and their operational conventions and restrictions.

Program Organization

This section explains the requirements for an executable Fortran program. It also describes the rules for ordering statements and the statement execution sequence.

Executable Programs

An executable program consists of exactly one main program and zero or more of each of the following entities

- function subprogram
- subroutine subprogram
- block data subprogram
- non−Fortran external procedure

The main program must not contain an **ENTRY** or a **RETURN** statement. On encountering a **RETURN** statement, the compiler issues a warning message; at execution time, a **RETURN** statement stops the program. Execution of a program normally ends when a **STOP** statement is executed in any program unit or when an **END** statement is executed in the main program.

Order of Statements

The following rules determine the order of statements in a main program or subprogram:

- In the main program, a **PROGRAM** statement is optional; if used, it must be the first statement. In other program units, a **FUNCTION**, **SUBROUTINE**, or **BLOCK DATA** statement must be the first statement.
- Comment lines can be interspersed with any statement and can precede a **PROGRAM**, **FUNCTION** , **SUBROUTINE**, or **BLOCK DATA** statement.
- **FORMAT** and **ENTRY** statements can be placed anywhere within a program unit after a **PROGRAM**, **FUNCTION**, **SUBROUTINE**, or **BLOCK DATA** statement.
- **ENTRY** statements can appear anywhere in a program unit except
  - between a block **IF** statement and its corresponding **END IF** statement
  - within the range of a **DO** loop that is, between a **DO** statement and the terminal statement of the **DO** loop
- The Fortran 77 standard requires that specification statements, including the **IMPLICIT** statement, be placed before all **DATA** statements, statement function statements, and executable statements.

  However, this implementation of Fortran permits the interspersing of **DATA** statements among specification statements.

  Specification statements specifying the type of symbolic name of a constant must appear before the **PARAMETER** statement that identifies the symbolic name with that constant.
- The Fortran 77 standard allows **PARAMETER** statements to intersperse with **IMPLICIT** statements or any other specification statements, but a **PARAMETER** statement must precede a **DATA** statement.

  This implementation extends the Fortran 77 standard to allow interspersing **DATA** statements among **PARAMETER** statements.

  PARAMETER statements that associate a symbolic name with a constant must precede all other statements containing that symbolic name.
- All statement function statements must precede the first executable statement.
- **IMPLICIT** statements must precede all other specification statements except **PARAMETER** statements.
- The last statement of a program unit must be an **END** statement.

Note: The above rules apply to the program statements after lines added by all **INCLUDE** statements are merged. **INCLUDE** statements can appear anywhere in a program unit.

Execution Sequence

The execution sequence in a Fortran program is the order in which statements are executed. Fortran statements are normally executed in the order they appear in a program unit. In general, the execution sequence is as follows:

1. Execution begins with the first executable statement in a main program and continues from there.
2. When an external procedure is referenced in a main program or in an external procedure, execution of the calling or referencing statement is suspended. Execution continues with the first executable statement in the called procedure immediately following the corresponding **FUNCTION**, **SUBROUTINE**, or **ENTRY** statement.
3. Execution is returned to the calling statement by an explicit or implicit return statement.
4. Normal execution proceeds from where it was suspended or from an alternate point in the calling program.
5. The executable program is normally terminated when the processor executes a **STOP** statement in any program unit or an **END** statement in the main program. Execution is also automatically terminated when an operational condition prevents further processing of the program.

The normal execution sequence can be altered by a Fortran statement that causes the normal sequence to be discontinued or causes execution to resume at a different position in the program unit. Statements that cause a transfer of control are

- **GO TO**
- arithmetic **IF**
- **RETURN**
- **STOP**
- an I/O statement containing an error specifier or end−of−file specifier
- **CALL** with an alternate return specifier
- a logical **IF** containing any of the above forms
- block **IF** and **ELSE IF**
- the last statement, if any, of an **IF** block or **ELSE IF** block
- **DO**
- terminal statement of a **DO** loop
- **END**