2 DATA TYPES AND OPERATIONS

We use computers to manipulate information that consists of letters, digits, and other special symbols. Such information is the interpretation of *data*. Although the word **data** is the plural of **datum**, many computer specialists use data as a mass noun such as *water* and *sand*. Data can be of different types. The basic data types in FORTRAN 77 are: **integer**, **real**, **character**, and **logical**. In this chapter we present these types in detail.

2.1 Constants

A constant is a fixed value of a data type that cannot be changed.

2.1.1 Integer Constants

Integer constants are whole numbers. An integer constant does not have a decimal point. Examples of integer constants are:

32 0 -6201 27 -83 1992

2.1.2 Real Constants

A real constant is a constant number that has a decimal point. Examples of real constants are 1.23, -0.0007, 3257.263, 5.0, 0.00002, 18., 774.00000, -64.9899 and 94000000000000000.0. The last number in the previous example leads us to the scientific notation for real numbers. 94000000000000000.0 can be written as 9.4×10^{16} or as 0.94×10^{17}. In FORTRAN, this number can be written in two possible ways: as 94000000000000000.0, or in scientific notation as 9.4E16 or 0.94E+17. Usually, such numbers are written in a way that the value of the first part is less than 1.0 and is greater than or equal to 0.1. The following table shows some examples of real numbers and their presentation in FORTRAN:

Real Number	Decimal Notation	FORTRAN Representation
6.3×10^{-5}	0.000063	0.63E-04
4.932×10^{7}	49320000.0	0.4932E+08
-5.7×10^{-6}	-0.0000057	-0.57E-05
5.7×10^{-6}	0.0000057	0.57E-05
5.7×10^{6}	5700000.0	0.57E+07

2.1.3 Logical Constants

There are two logical constants; *true* and *false*. In FORTRAN, the logical constant *true* is written as **.TRUE.** and the logical constant *false* is written as **.FALSE.**.

2.1.4 Character Constants

FORTRAN allows character usage and manipulation. Character constants must be placed between two consecutive single quotes. A character constant is also referred to as a character string. The following table shows some character constants and their representation in FORTRAN:

Character Constant	FORTRAN Representation
THIS IS CHAPTER TWO	'THIS IS CHAPTER TWO'
MORE THAN ONE BLANK	'MORE THAN ONE BLANK'
ISN'T IT?	'ISN''T IT?'
1234 AS CHARACTERS	'1234 AS CHARACTERS'

Note that if a single quote needs to be included in a character constant, it should be written as two single quotes.

2.2 Variables

A variable is an object of a certain data type that takes a value of that type. A variable, as the name suggests, can change its value through certain FORTRAN statements such as the assignment statement (section 2.5) and the **READ** statement (section 2.6). When a variable is defined, the compiler allocates specific memory location to that variable. This location must be given a name to be referenced later. The name of such a location is called a *variable name*. We shall use the term *variable* to mean *variable name*. Before using a variable we may define it. The definition of a variable means that we are allocating a memory location for that variable. However, it does *not* mean that the compiler assigns a value to the variable. There are some rules for choosing variable names in FORTRAN. These rules are as follows:

- The variable should start with an alphabetic character (A, B, C,...,Z)
- The length of the variable should not exceed 6 characters.
- A variable may contain digits (0, 1, 2, ..., 9).
- A variable should not contain special characters ($, ;, ,, :, !, ~, ^,(,{, [,), },], <, >, ?, “, ‘, \, | , @, %, &, #, +, -,/,*, .., etc.).
- A variable should not contain blanks.

Examples of valid and invalid variable names are given below:

Variable	Comment
TRY	Valid.
NAME21	Valid.
NAME211	Invalid. Length is more than 6 characters.
A+B	Invalid. Special character '+' can not be used.
5TEST	Invalid. Name does not start with a letter.
FIVE7	Valid.

The following subsections present different variable types and how to define them.

2.2.1 Integer Variables

Integer variables can hold only integer values. There are two ways to define an integer variable in FORTRAN: *explicitly* and *implicitly*. The *explicit* definition allows us to define variable types, irrespective of the first letter of the variable name. In such a case, we must use the **INTEGER** statement. The general form of this statement is as follows:

INTEGER *list of integer variables*

where *list of integer variables* is a list that has the names of variables separated by commas. The **INTEGER** statement is a FORTRAN declaration statement. This statement must be typed starting in either column 7 or after and must appear at the beginning of the program before any other executable statement. In fact, all declaration statements must appear at the beginning of the program. The following examples demonstrate the use of the **INTEGER** statement:

Example	Comments
`INTEGER BOOKS, NUM, X`	Three integer variables: BOOKS, NUM, X
`INTEGER Y1, AB3W`	Two integer variables: Y1, AB3W
`INTEGER CLASS, ID, TOTAL`	Three integer variables: CLASS, ID, TOTAL
`INTEGER SUM`	One integer variable: SUM

It is a good programming habit to use *explicit* definition in writing their programs. This minimizes logical errors that may arise while running such programs.

In *implicit* definition, we choose a variable name that starts with one of the following letters: **I, J, K, L, M, N.** Hence, any variable that starts with one of these letters is considered implicitly as an integer variable unless it is otherwise explicitly stated. Examples of integer variables are:

NUMB, N1, LAB, ISUM, JX, KILO, MEMO.

Implicit definition is assumed when a programmer forgets to use explicit definition.

2.2.2 Real Variables

Real variables can hold only real values. As was the case in integer variable definition, there are two ways to define a real variable: *explicitly* and *implicitly*. The explicit definition allows us to define variable types irrespective of the first letter of the variable name, using the **REAL** statement. The general form of this statement is as follows:

REAL *list of real variables*

where *list of real variable* is a list that has the names of variables separated by commas. The **REAL** statement is a FORTRAN declaration statement. It must be typed starting in either column 7 or after and must appear in the beginning of the program before any other executable statement. The following examples demonstrate the use of the **REAL** statement:

Example	Comments
`REAL NOTES, NUM2, IX`	Three real variables: NOTES, NUM2, IX
`REAL M1, AB3`	Two real variables: M1, AB3
`REAL INSIDE, KD2, SBTOT`	Three real variables: INSIDE, KD2, SBTOT
`REAL J1SUM`	One real variable: J1SUM

We should try our best to declare our variables explicitly. If we forget to use explicit definition, then FORTRAN compilers assume implicit definition.

In *implicit* definition, any variable that does not start with one of the letters **I, J, K, L, M, N** is considered, implicitly, as a real variable unless the type of the variable is explicitly stated. Examples of real variables are:

YNUMB, X1, PERC, SUM, RJX, TOTAL, STID, A5, EPSLON, PI.

2.2.3 Logical Variables

Logical variables have either a **.TRUE.** or a **.FALSE.** value. There is only one way to define logical variables - they must be declared explicitly. The statement that is used to define logical variables is the declarative **LOGICAL** statement. This statement should be typed starting either in column 7 or after. It must appear at the beginning of the program before any executable statement. The general structure of the **LOGICAL** statement is:

LOGICAL *list of logical variables*

where *list of logical variables* is one or more variables separated by commas. Examples of **LOGICAL** statement usage are given below:

Example	Comments
`LOGICAL TEST, FLAG, Q, P`	Four logical variables: TEST, FLAG, Q, P
`LOGICAL M5`	One logical variable: M5
`LOGICAL SORTED, LINK`	Two logical variables: SORTED, LINK

2.2.4 Character Variables

Character variables must be given character constants as their values. Only explicit definition allows us to define character variables. The declaration statement that is used in character definition is the **CHARACTER** statement. As is the case in other types of declaration statements, the **CHARACTER** declaration statement must appear at the beginning of the program and should be typed before any executable statement. The general form of the **CHARACTER** statement is as follows:

CHARACTER *list of character variables with their lengths*

or

CHARACTER*n *list of character variables with their lengths*

where *list of character variables with their lengths* consists of one or more variables separated by commas. Each variable may be followed by ***k**, where **k** is a positive integer specifying the length of the string that particular variable can hold. If ***k** is not specified, the length of that variable is assumed to be **n**. If **n** is not specified, the length is assumed to be 1. The following table shows some examples of **CHARACTER** statements.

Example	Character variables and their lengths
`CHARACTER NAME*20`	NAME is a character variable of length 20
`CHARACTER*6 M, WS*3, IN2`	M and IN2 are of length 6; WS is of length 3
`CHARACTER T1, T2, T3`	T1, T2 and T3 are of length 1
`CHARACTER Z*8, TEST`	Z is of length 8 and TEST is of length 1
`CHARACTER*12 Z1, Z2`	Z1 and Z2 are of length 12

Detailed character manipulation and usage will be discussed in chapter 10. In the remainder of this chapter, we present arithmetic and logical operations, the assignment statement, and simple input/output statements.

2.3 Arithmetic Operations

Addition, subtraction, multiplication, division, and exponentiation (power) are called arithmetic operations. The following subsections present details about these operations.

2.3.1 Arithmetic Operators

In FORTRAN there are five basic operators. These operators are shown in the following table with the sequence in which they are evaluated (precedency):

FORTRAN Operator	Operation	FORTRAN Example	Math Notation	Precedency
**	Exponentiation	`X ** Y`	x^y	1
*	Multiplication	`X * Y`	$x \times y$	2
/	Division	`X / Y`	$x \div y$	2
+	Addition	`X + Y`	$x + y$	3
-	Subtraction	`X - Y`	x - y	3

An arithmetic expression consists of one or more arithmetic operations. Operations that are applied on two operands are called binary operations. Operations that are applied on one operand are called unary operations. The minus operator '-' may be used as a unary operator or as a binary one. An operand can be a constant value, a variable that has been given a value, or a correct expression.

In any arithmetic expression, parentheses have the highest priority (precedence) in evaluation. In the case of nested parentheses (parentheses inside parentheses), evaluation starts with the most-inner parentheses. The next higher priority operator is

the exponentiation (also called power) operator '**'. If there are two or more consecutive exponentiation operators in an arithmetic expression, evaluation of these exponentiation operations is done from right to left. For example, in the expression 2**2**3, we start evaluating 2**3 (which is 8) and after that we evaluate 2**8 (which is 256). Division and multiplication operators have the same priority, but they are lower in priority than the exponentiation operator. The addition and subtraction operators have the same priority which is lower than the priority of multiplication and division operators. Operators with the same priority are evaluated from left to right with the exception of the exponentiation operator as explained earlier.

There are two restrictions on the use of arithmetic operators. The first restriction is that no two operators must appear consecutively. For example, if the expression 2 * -3 is intended, in FORTRAN, it should be written as 2*(-3). The second restriction is on the use of the exponentiation operator. This operator must not be used to raise a negative number to a real exponent. For example, expressions such as (-2.0) ** 1.5 or (-3) ** 2.3 are not allowed in FORTRAN language. To compute x^y, when y is real, most FORTRAN Compilers use the mathematical formula $e^{y \ln x}$. When x is negative, the value of $\ln x$ is undefined.

2.3.2 Integer Operations

An operator between two integer operands is considered to be an integer operator and the operation is considered to be an integer operation. Integer operations always produce integer results. The fraction part is ignored. The following table shows some examples of integer operations:

Expression	Value	Comment
50 - 23	27	
3 ** 2	9	
5 * 7	35	
8 / 2	4	
8 / 3	2	Fraction part is truncated (not 2.6666667)
9 / 10	0	Fraction part is truncated (not 0.9)

Note that the expression I/J * J is not always equivalent to I. For example, if I and J are integer variables, and the value of I is 17 and the value of J is 6, the expression becomes 17 / 6 * 6. To evaluate this expression we consider operator precedence. Since operators '/' and '*' have the same priority, they are evaluated from left to right. We start with 17 / 6. The two operands are integers and therefore '/' here is an integer operator. The result must be an integer, which in this case evaluates to 2. Now, evaluation proceeds as 2 * 6 which results in 12 and not 17.

2.3.3 Real Operations

An operator between two real operands is considered to be a real operator and the operation is considered to be a real operation. Real operations produce real results. The following table shows some examples of real operations:

Expression	Value
50.0 - 23.0	27.0000000
3.0 ** 2.0	9.0000000
5.0 * 7.0	35.0000000
8.0 / 2.0	4.0000000
8. / 3.0	2.6666667
9. / 10.	0.9000000
9.3 / 3.2	2.9062500

2.3.4 Mixed-mode Operations

An operator between an integer operand and a real operand is considered to be a mixed-mode operator and the operation is considered to be a mixed-mode operation. Mixed-mode operations produce real results. The following table shows examples of mixed-mode operations:

Expression	Value	Comment
50 - 23.0	27.0000000	
3.0 ** 2	9.0000000	
3 ** 2.0	9.0000000	
4** 0.5	2.0000000	
5.0 * 7	35.0000000	
56.7 / 7	8.1000000	
8 / 2.0	4.0000000	
8.0 / 3	2.6666667	
9 / 10.	0.9000000	Decimal point can be placed without zero.
17 / 6 * 6.0	12.0000000	'/' is an integer operator and '*' is a mixed mode operator

The number of positions to the right of the decimal point in a real number depends on the computer used. In the examples above, we have assumed that the computer allows up to 7 positions.

2.3.5 Examples

Example 1: *Evaluate the following arithmetic expression*

20 - 14 / 5 * 2 ** 2 ** 3

Solution:

Expression: 20 - 14 / 5 * 2 ** 2 ** 3
Priority is for ** from right to left
Step 1: 2 ** 3 = 8 (integer operation)
Expression: 20 - 14 / 5 * 2 ** 8
Priority is for ** from right to left
Step 2: 2 ** 8 = 256 (integer operation)
Expression: 20 - 14 / 5 * 256

Priority is for / and * from left to right
Step 3: 14 / 5 = 2 (integer operation)
Expression: 20 - 2* 256
Priority is for *
Step 4: 2 * 256 = 512 (integer operation)
Expression: 20 - 512
Priority is for -
Result: -492

Example 2: *Evaluate the following arithmetic expression*

14.0 / 5 * (2 * (7 - 4) / 4) ** 2

Solution:

Expression: 14.0 / 5 * (2 * (7 - 4) / 4) ** 2
Priority is for expression inside the inner most parenthesis
Step 1: (7 - 4) = 3 (integer operation)
Expression: 14.0 / 5 * (2 * 3 / 4) ** 2
Priority is for expression inside the parenthesis
Step 2 & 3: (2 * 3 / 4) = (6 / 4) = 1 (2 integer operations)
Expression: 14.0 / 5 * 1 ** 2
Priority is for **
Step 4: 1 ** 2 = 1 (integer operation)
Expression: 14.0 / 5 * 1
Priority is for / and * from left to right
Step 5: 14.0 / 5 = 2.8000000 (Mixed mode operation)
Expression: 2.8000000 * 1
Priority is for *
Result: 2.8000000

Example 3: *Rewrite the following FORTRAN expression as a mathematical form*

X + Y / W - Z

Solution:

$$x + \frac{y}{w} - z$$

Example 4: *Rewrite the following FORTRAN expression as a mathematical form*

X ** (1.0 / 2.0) / Y ** Z

Solution:

$$\frac{\sqrt{x}}{y^z} \quad or \quad \frac{x^{\frac{1}{2}}}{y^z}$$

Example 5: *Convert the following mathematical expression into FORTRAN expression. Use minimum number of parenthesis*

$$\frac{\sqrt{a+b}}{a^2 - b^2}$$

Solution:

(A + B) ** 0.5 / (A ** 2.0 - B ** 2.0)

2.4 Logical Operations

Logical operations evaluate to either **.TRUE.** or **.FALSE.**. The following subsections discuss logical operators, relational operators and logical expressions:

2.4.1 Logical Operators

This section discusses the three logical operators: **.AND., .OR.** and **.NOT.**. The **.AND.** operator is a binary logical operator that produces .TRUE., if and only if, both its operands have a .TRUE. value. If any of the operands have a .FALSE. value, the result of the operation is .FALSE.. The **.OR.** operator is a binary logical operator that produces .FALSE. if and only if both operands have the value .FALSE.. otherwise, the result is .TRUE.. The unary logical operator **.NOT.** produces the opposite value of its operand. The following table shows the results of the three logical operations .AND., .OR. and .NOT. on different operand values, assuming P and Q are logical variables:

P	Q	P .AND. Q	P. OR. Q	.NOT. P
.FALSE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.
.TRUE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.

The *.NOT. operator has the highest priority of the three logical* operators followed by the *.AND.* operator. The *.OR. operator has the lowest priority.* These operators are shown in the following table with the sequence in which they are evaluated (precedency):

Logical Operator	FORTRAN Example	Precedence
.NOT.	.NOT. P	1
.AND.	P .AND. Q	2
.OR.	P .OR. Q	3

Example 1: *Evaluate the following logical expression:*

.FALSE. .OR. .NOT. .TRUE. .AND. .TRUE.

Solution:

Expression: .FALSE. .OR. .NOT. .TRUE. .AND. .TRUE.

priority is for .NOT.

Step 1: .NOT. .TRUE. is .FALSE.

Expression: .FALSE. .OR. .FALSE. .AND. .TRUE.

priority is for .AND.

Step 2: .FALSE. .AND. .TRUE. is .FALSE.

Expression: .FALSE. .OR. .FALSE.

priority is for .OR.

Result: .FALSE.

Example 2: *Assume that the following declaration is given:*

LOGICAL FLAG

If it is known that the expression

.NOT. FLAG .OR. .FALSE.

has the value .TRUE., what is the value of FLAG?

Solution:

The final result must be .TRUE.. The last step is *somevalue* .OR. .FALSE. because the .NOT. operator has higher priority than the .OR. operator. *somevalue* .OR. .FALSE. will have the value .TRUE. if and only if the value of *somevalue* is .TRUE.. But *somevalue* is equivalent to .NOT. FLAG, therefore the value of FLAG is .FALSE..

2.4.2 Relational Operators

The values of arithmetic expressions can be compared using relational operators. The following table shows the different relational operators. Assume all variables have been initialized:

Operator	Math	Example	Description
.EQ.	=	`X .EQ. Y`	True if X and Y are equal
.NE.	≠	`N .NE. 8`	True if N is not equal to 8
.GT.	>	`P1 .GT. 7.3`	True if P1 is greater than 7.3
.GE.	≥	`SM .GE. TOT`	True if SM is greater than or equal to TOT
.LT.	<	`A+B.LT.A*2.0`	True if the sum of A and B is less than 2A
.LE.	≤	`NUM.LE.CLASS`	True if NUM is less than or equal to CLASS

A relational expression evaluates to either .TRUE. or .FALSE.. Relational operators have lower priority than arithmetic operators and higher priority than logical operators. They are evaluated from left to right. The next subsection presents the use of relational, logical, and arithmetic operators in logical expressions.

2.4.3 Logical Expressions

A logical expression evaluates to .TRUE. or .FALSE.. It may contain different types of variables and operators. It may contain arithmetic expressions, logical expressions, and relational expressions. Logical expressions are used in selection constructs which are discussed in chapter 3. The evaluation of a logical expression starts with the evaluation of arithmetic expressions first followed by the relational expressions, and finally the logical expressions. The following examples demonstrate the evaluation of logical expressions:

Example 1: *Given that X has a value of 3.0, Y has a value of 5.0, Z has a value of 10.0, and FLAG is a logical variable with .FALSE. value, evaluate the following FORTRAN expression:*

.NOT. FLAG .AND. X*Y .GT. Z .OR. X+Y .GT. Z

Solution:

Expression: .NOT. FLAG .AND. X*Y .GT. Z .OR. X+Y .GT. Z

Evaluate arithmetic expressions first.

Expression: .NOT. FLAG .AND. 15.0 .GT.10.0 .OR. 8.0 .GT.10.0
Evaluate relational expressions next.
Expression: .NOT. FLAG .AND. .TRUE. .OR. .FALSE.
Evaluate logical expressions. Start with .NOT..
Expression: .TRUE. .AND. .TRUE. .OR. .FALSE.
Evaluate logical .AND. next.
Expression: .TRUE. .OR. .FALSE.
Evaluate .OR. next
Result: .TRUE.

Example 2: *When is the value of the following expression .TRUE.? Assume K and L are integers.*

K / L * L .EQ. K

Solution:

If K is divisible by L, the value of the expression is .TRUE.. Otherwise, the value will be .FALSE..

Example 3: *Given that X has a value of 3.0, Y has a value of 5.0, Z has a value of 10.0, and FLAG is a logical variable with the value .FALSE., find the value of each of the following expressions:*

.NOT. FLAG .OR. FLAG
X .GT. Y - Z / 2.0
X*Z .EQ. 20.0 .OR. FLAG .AND. .NOT. Z .EQ. 5.0
X .GT. Y .AND. X .GT. Z .OR. X .LT. Y .AND. X .LT. Z
Z*10 .NE. Y*30 .AND. X .LE. Y .AND. FLAG
.NOT. FLAG .AND. FLAG
.NOT. .NOT. FLAG

Solution:

Expression	**Value**
`.NOT. FLAG .OR. FLAG`	.TRUE.
`X .GT. Y - Z / 2.0`	.TRUE.
`X*Z .EQ. 20.0 .OR. FLAG .AND. .NOT. Z .EQ. 5.0`	.FALSE.
`X .GT. Y .AND. X .GT. Z .OR. X .LT. Y .AND. X .LT. Z`	.TRUE.
`Z*10 .NE. Y*30 .AND. X .LE. Y .AND. FLAG`	.FALSE.
`.NOT. FLAG .AND. FLAG`	.FALSE.
`.NOT. .NOT. FLAG`	.FALSE.

2.5 Assignment Statement

The assignment statement in FORTRAN assigns a value to a variable. The general form of the FORTRAN assignment statement is:

variable = *expression*

where *expression* must have a value of the same type as the *variable* with one exception: integer values can be assigned to real variables and real values can be assigned to integer variables. In assigning a real value to an integer variable, the decimal part is truncated before the value is stored in the variable. In the case of an integer value

being assigned to a real variable, the integer value is converted to a real value before it is stored in the variable. The FORTRAN assignment statement is **not a mathematical equation**. Therefore, it is possible to write assignment statements such as:

```
      X = 1.0
      X = X + 1.0
```

where the first statement assigns the value 1.0 to the variable X. The second statement evaluates the expression X + 1.0 which will be 2.0 and then assigns the result to the variable X. It should be clear that the old value of X (i.e 1.0) is changed to the new value (i.e. 2.0).

Example 1: *Write FORTRAN assignment statements to store the real number 3.25 into the variable X1 and 7.0 into the variable Y1.*

Solution:

```
      X1 = 3.25
      Y1 = 7.0
```

Example 2: *Write a FORTRAN assignment statement to store in X1 the value stored in Y1.*

Solution:

```
      X1 = Y1
```

Example 3: *Write a FORTRAN assignment statement to increment X1 by 1.*

Solution:

```
      X1 = X1 + 1.0
```

Example 4: *Write a FORTRAN assignment statement to add to X1 the value of Y1.*

Solution:

```
      X1 = X1 + Y1
```

Example 5: *Write a FORTRAN assignment statement to store in X1 the contents of X1 times the contents of Y1.*

Solution:

```
      X1 = X1 * Y1
```

Example 6: *Assume that the coefficients of a quadratic equation are given as A, B, and C. Write FORTRAN assignment statements to find the two roots, ROOT1 and ROOT2, of the quadratic equation.*

Solution:

```
      ROOT1 = (-B + (B ** 2.0 - 4.0 * A * C) ** 0.5) / (2.0 * A)
      ROOT2 = (-B - (B ** 2.0 - 4.0 * A * C) ** 0.5) / (2.0 * A)
```

Example 7: *Given SUM as the sum of student grades in an exam and COUNT as the number of students, write an assignment statement to find the average AVER.*

Solution:

```
      AVER = SUM / COUNT
```

Example 8: *Write FORTRAN assignment statements to exchange the values of the variables X and Y. (Hint: Use a temporary variable T)*

Solution:

```
T = X
X = Y
Y = T
```

Example 9: *If the variable **NAME** is declared as follows:*

```
CHARACTER NAME * 8
```

*what will the value of **NAME** be after the following assignment statement is executed?*
NAME = 'ICS101 FORTRAN'

Solution:

Since the length of the variable **NAME** is declared as 8, the assignment statement will assign the first 8 characters of the string constant to **NAME**. Hence, the value of **NAME** is going to be:

ICS101 F

Example 10: *Given the following declaration and assignment statements:*

```
CHARACTER MAJOR * 15
MAJOR = 'FINAL'
```

what is the value of the variable **MAJOR** ?

Solution:

Since the length of the variable **NAME** is declared as 15, the assignment statement will assign the string constant FINAL to the first 5 positions of **MAJOR** and fill the remaining 10 positions with blanks.

2.6 Simple Input Statement

We may assign a value to a variable by using either the assignment statement or by reading an input value into the variable. To read an input value from the terminal into a variable, we must use an input statement. There are two types of input statements: the formatted **READ** and the unformatted **READ**. This section presents the unformatted **READ** statement. The general form of the unformatted **READ** is

READ*, *list of variables separated by commas*

The following points must be noted while using the unformatted **READ** statement:

- Each read statement starts reading from a new line.
- If the input data is not enough in the current line, reading continues in the next line.
- The data values can be separated by blanks or comma.
- The data values must agree in type with the variables.
- Integer values can be read into real variables but real values must not be read into integer variables.
- Extra data on an input line is ignored.

2.6.1 Examples

Example 1: *Assume the following declaration:*

```
INTEGER NUM, M1, K, L1, L2, L3, K1, K2
REAL TOT, X1, YY, S, ST, A, X, Y, Z
```

*The following table gives examples of **READ** statements:*

Statement	Input Line	Effect
READ*, NUM, TOT	9 5.08	NUM = 9 TOT = 5.08
READ*, X1, YY	325 27	X1 = 325.0 YY = 27.0
READ*, M1	20.0	ERROR MESSAGE. DATA TYPE MISMATCH
READ*, K, S	18, 0.35E-2	K = 18 S = 0.35E-2
READ*, ST	-23.4	ST = -23.4
READ*, L1, L2, L3	7 6 5	L1 = 7 L2 = 6 L3 = 5
READ*, A, A	1.0, 2.0	A = 2.0
READ*, K1 **READ***, K2	5 8 20 9	K1 = 5 K2 = 20
READ*, X, Y, Z	5 8 20 9	X = 5.0 Y = 8.0 Z = 20.0

Example 2: *Assume the following declaration:*

```
CHARACTER NAME*9, STR1*5, STR2*3
LOGICAL P1, P2
```

*The following table gives examples of **READ** statements:*

Statement	Input Line	Effect
READ*, NAME	'AHMED ALI'	NAME = 'AHMED ALI'
READ*,STR1, STR2	'ALI' 'CLASS'	STR1 = 'ALI ' STR2 = 'CLA'
READ*, P1, P2	T F	P1 = .TRUE. P2 = .FALSE.

2.7 Simple Output Statement

The **PRINT** output statement is used to print the values of variables, expressions or constants. There are two types of **PRINT** output statements: the formatted **PRINT** statement and the unformatted **PRINT** statement. The formatted **PRINT** statement will be discussed in chapter 8. The general form of the unformatted **PRINT** statement in FORTRAN is

PRINT*, list of variables, expressions, or constants separated by commas

The following subsection presents some examples on **PRINT** statement.

2.7.1 Examples

Example 1: *In the table below, examples of the **PRINT** statement are given assuming the following initializations:*

```
      LOGICAL FLAG
      INTEGER K, L
      REAL S1, S2
      FLAG = .TRUE.
      K = 3
      L = 20
      S1 = 35.0
      S2 = S1 - K - L
```

Statement	Output	Comments
`PRINT*, K, S1`	`3 35.0000000`	Blanks depends the type of computer
`PRINT*, L+S2, W`	`32.0000000 ???????`	??????? for undefined
`PRINT*, L, FLAG`	`20 T`	
`PRINT*, L / K * K`	`18`	
`PRINT*, L / K * K * 1.0`	`18.0000000`	
`PRINT*, L * 1.0 / K * K`	`20.0000000`	May be 19.9999994 (accuracy)
`PRINT*,5,6+7, L, 2, K+3`	`5 13 20 2 6`	Constants and expressions
`PRINT*, 'K= ',K,' L IS ',L`	`K= 3 L IS 20`	Characters may be printed
`PRINT*, 'THIS TESTS'`	`THIS TESTS`	
`PRINT*, FLAG, .FALSE.`	`T F`	Logical values either T or F
`PRINT*`		Prints an empty line

Example 2: *In the table below, more examples of the* ***PRINT*** *statement are given assuming the following initializations:*

```
      CHARACTER*10 LSTNAM
      CHARACTER CLASS*5, MAJOR*4
      LSTNAM = 'AL-FORTRAN'
      CLASS = 'BATAL'
      MAJOR = 'ANY1'
```

Statement	Output	Comments
`PRINT*, CLASS, MAJOR`	`BATALANY1`	No blanks in between
`PRINT*,LSTNAM, ' ',MAJOR`	`AL-FORTRAN ANY1`	Explicit blank as it is

The following points must be noted while using the **PRINT** statement:

- Each **PRINT** statement starts printing on a new line.
- If the spaces in the line are not enough to hold the whole output, printing continues on the next line.
- A variable that does not have a value will produce question marks if it is printed.

2.8 A Complete Program

The following program reads three real numbers, prints them, computes their average and prints it:

```
C      THIS PROGRAM READS 3 REAL NUMBERS
C      AND COMPUTES AND PRINTS THE AVERAGE
C
       REAL NUM1, NUM2, NUM3, COUNT, AVER
       COUNT = 3.0
       READ*, NUM1, NUM2, NUM3
       PRINT*, 'THE NUMBERS ARE ', NUM1, NUM2, NUM3
       AVER = (NUM1 + NUM2 + NUM3) / COUNT
       PRINT*, 'THE AVERAGE IS ', AVER
       END
```

The first three lines are comment lines. We can insert comment lines anywhere in the program. Each comment line must start with 'C' or '*' in column one. The fourth statement of the program is the **REAL** declaration statement. It declares five real variables that are going to be used in the program. The next statement is an assignment statement that assigns 3.0 to the variable COUNT. The **READ** statement will read 3 values from the input line and assign them to the variables NUM1, NUM2, and NUM3, respectively. The first **PRINT** statement is used to print the values that were read. The next statement is an assignment statement that computes the average. The result is stored in the variable AVER. The second **PRINT** statement prints the average with a proper message. The last statement is the **END** statement. The **END** statement signals the physical end of the program.

If the input line of this program is

```
9.0   8.0   10.0
```

the output is as follows:

```
THE NUMBERS ARE 9.0000000 8.0000000 10.0000000
THE AVERAGE IS 9.0000000
```

In FORTRAN programs, execution starts from the beginning of the program and proceeds statement by statement, in sequence, unless there is an indication for changing the sequence. Statements that may change the sequence of execution are selection and repetition statements. Selection is discussed in chapter 3 and repetition in chapter 5.

2.9 Exercises

1. Evaluate the following arithmetic expressions:
   1. 4 ** 2 / 3
   2. ((2 + 6) / 2 + 3.0 /6.0 *4) * (2 / 4)
   3. 10 ** 2 ** 3
   4. 10 / 4 /4 + (2 - 10 / 2.0)
2. Indicate if the statements below are valid FORTRAN statements or not:
   1. Y + X = K
   2. AB = A * B
   3. **PRINT***, 1.0, '+', 2.0, '=', 1.0 + 2.0
   4. X = Y ** -3
   5. X12345 = 8.0
   6. X = Y = 5.0
   7. P = (Q + R) * (- (-8))

8. X3X = 8.0
9. **READ***, R+A
10. **READ***, NUM,NUM

3. What will be printed by the following FORTRAN 77 programs ?

```
1.      INTEGER I, J, K
        I = 300
        J = 500
        K = J/I
        PRINT*, K
        END
```

```
2.      INTEGER ONE,TWO,THREE,FOUR,FIVE
        ONE = 1
        TWO = 2
        THREE = 3
        FOUR = 4
        FIVE = THREE + FOUR ** ( ONE / TWO )
        PRINT*, FIVE
        END
```

```
3.      INTEGER  M, N
        READ*, M
        READ*, N
        PRINT*, M, N
        END
```

Assume the input for the program is:

```
7   9
```

```
4.      INTEGER I, J, K, L
        READ*, I, J
        READ*, K, I
        PRINT*, I, J, K, L
        END
```

Assume the input for the program is:

```
4   5   6
7   8   9
```

```
5.      REAL X
        X = 1.2
        X = X + 1.0
        X = X + 1.0
        X = X + 1.0
        PRINT*, X , X, X, X
        END
```

```
6.      REAL A, X
        A = 8 ** 1/3
        X = 25 ** 1/2
        PRINT*, X, A
        END
```

```
7.      INTEGER XLM, NUM1, NUM2
        REAL PNM
        READ*, NUM1, NUM2
        PNM = NUM1 / NUM2
        XLM = 3 / PNM * 3.00 ** NUM2
        PRINT*, PNM, NUM1, NUM2, XLM
        END
```

Assume the input for the program is:

```
3,2
```

4. What is the value of each of the following expressions? Use the following values if needed:

```
        REAL A, B
        INTEGER K, J
        A = 2.0
        K = -2
        B = 3.5
        J = 1
```

1. 6 * J / K * 4
2. 9 + K / 5 * A / 2
3. A / (B + K) / J
4. 3 ** J ** A ** 1 + K / J
5. -2 / 4 * 4 ** 2
6. -2 / 4.0 * 2 ** 2 + 2 * 4.0 ** 2
7. 3 ** 2.0 * (3.0 - 1) + 2.0 * 1 * 3.0
8. 5 ** 3 / 2 ** 5 / 2
9. (5 / 2) ** 1.0 ** 2
10. (1 + (3.2 * 2 - (5 - 4)))
11. ((2 + 6) / 2 + 3.0 / 6.0 * 4) * (2 / 4)
12. 99999 / 100000 - 1
13. 2 ** 2 ** 3
14. 9 / 4 * 2 ** 1 / 2
15. 900 / 3.0E2

5. Convert the following FORTRAN assignment statements into an algebraic form :
1. W = (X / Y / Z * T) ** 3 + 1 + 1.674E-24 * C
2. Q = 1012.0 * P ** 0.5 * (1.0 - P / 100.0)
3. K = A * B / C - 2

6. Which of the following are valid FORTRAN variable names?
1. CS101GRADE
2. AH/Q
3. PRICE
4. +RATE
5. 2THIRD
6. NUMB12
7. IDNUMB

8. WHOLE-SALE-PRICE
9. $FORT
10. Y8X
11. ALL*

7. Indicate the following statements as either TRUE or FALSE:
   1. A **REAL** statement is an executable statement.
   2. Compiling the statement Y = 2 ** 4 ** 3.5E50 will cause syntax error.
   3. The statement **INTEGER** X,Y,Z implies that XYZ is an integer variable.
   4. If J, K, and L are integers, then the FORTRAN expressions (J + K) / L and (J / L) +(K / L) are equivalent.
   5. The **INTEGER** statement can appear any where in the program.
   6. If K and L are integers, then the FORTRAN expressions K * L**2 / K**2 and K * (L**2 / K**2) are equivalent.
   7. **PRINT***,X=5 is a valid FORTRAN 77 statement.
8. Add the minimum number of parentheses to the FORTRAN expression

   A ** B ** 2 + B - C / D + A * B / C * D

   to be equivalent to the mathematical expression :

$$\frac{a^{(b)^{2+b-c}}}{d+a} \times \frac{b}{cd}$$

9. In the following FORTRAN expression the operators have been numbered :

   1 2 3 4 5 6 7 8 9

   A ** B ** 2 + B - C / D + A * B / C * D

   Give the order in which the operators are evaluated a cording to FORTRAN 77 rules. (only write the operator numbers in order)

10. Write a FORTRAN program to read a 3 digit number, then prints the hundredth, the tenth, and the ones digits. If the input is:

```
728
```

The output should be:

```
THE HUNDREDS DIGIT =  7
THE TENTH DIGIT    =  2
THE ONES DIGIT     =  8
```

11. Write a FORTRAN program which reads the radius of a sphere and calculates the surface area and the volume of the sphere. Your program should print the radius, surface area and the volume:

    Surface area = $4\pi r^2$

    Volume = $\frac{4}{3}\pi r^3$

12. Convert the following mathematical expressions / assignments to FORTRAN expressions / assignments. (do not use extra parentheses)

    1. $2x + \frac{y}{2}$

2. $\sqrt{\frac{a+b}{a-b}}$
3. $\frac{r^2}{3} - \frac{ac^{\frac{3}{4}}}{2b}$
4. $\frac{1}{\frac{1}{r1}+\frac{1}{r2}+\frac{1}{r3}}$
5. $a = b + \frac{xy}{c+d} + 2$
6. $2a + c^{-6}$
7. $\frac{a+\sqrt[4]{b}}{\frac{2}{a^2+5}} - 1$

13. For each of the following FORTRAN expressions, write an equivalent expression by deleting all "REDUNDANT" parentheses (i.e. parentheses whose deletion does not change the result of the expression).
    1. (A*B) * C / ((X*Y) **2)
    2. ((A+B) ** 2 + (3*C) ** 3) ** (A/B)
    3. ((A-B) +C) +(D*E)
    4. (C*X) ** ((2-A) * B)
    5. -B + ((B**2 - (4 * (A*C)))) ** 0.05

14. Write a program that converts a quantity expressed in seconds to a correspondence quantity expressed in hours, minutes and seconds. If the input is:

```
8125
```

The output should be:

```
2 HOURS, 15 MINUTES, 25 SECONDS.
```

15. The input data to a certain program is more than what is required. The data is as follows:

```
4   5   12   10
6   1   8   13   19
3   2   9   0   7   18   20
```

Write a FORTRAN program to read enough data (i.e. using the minimum number of variables in the **READ** statement) to print the following output:

```
4 5
1 8
9 0
```

(your program should have **READ** and **PRINT** statements only)

16. i) The output of the program below is as follows:

```
8
```

Fill in the spaces to get the output shown above

```
      INTEGER K, M, N
      K = -----
      M = 2
      N = 3
      PRINT*, M**N**M**K
      END
```

ii) The output of the program below is as follows:

```
1 4
7 8 10
```

Fill in the spaces to get the output shown above

```
      INTEGER K1, K2, K3, K4, K5
      READ*, -------
      READ*, -------
      READ*, -------
      PRINT*, K1, K2
      PRINT*, K3,K4,K5
      END
```

Assume the input for the program is:

```
1 2 3
4 5 6
7 8
10 11 12
```

17. Determine whether the following conditions are TRUE or FALSE. Assume A = 3.5, B = -4.1, I = -4, J = 9, FLAG = .TRUE. when needed:
    1. (3.0/2.LT.1.5).AND.(4/2.GT.1)
    2. .FALSE..AND..TRUE..OR..NOT.(.FALSE..AND..TRUE.)
    3. .NOT..FALSE..AND..TRUE.
    4. .NOT..FALSE..OR..TRUE..AND.3/2.EQ.1.0
    5. .NOT.5**2.EQ.5*2.AND.0.GT.5.OR.5*2+2.GT.0
    6. A.GT.B.OR.I.EQ.J.AND.FLAG
    7. A+I-4.GT.B-3+2*J.OR.A*B.GT.2.0*I
    8. FLAG.OR.(A-I)/(B-J).GT.1.021
    9. .NOT.(A.GT.B).OR.(I.GT.J)
    10. (A+B)/(I+J).LT.-5.0.AND..NOT.A*I.LE-.14.0
    11. .NOT.(.NOT.FALSE.).AND..TRUE..OR..FALSE.

2.10 Solutions to Exercises

Ans 1

1. 5 2. 0.0 3. 100000000 4. -3.0

Ans 2.

1. Invalid 2. Valid 3. Valid 4. Invalid 5. Valid
6. Invalid 7. Valid 8. Valid 9. Invalid 10. Valid

Ans 3.

1
4

Error Message
8 5 7 ???????
4.2 4.2 4.2 4.2
12.0 2.0
1.0 3 2 27

Ans 4.

1. -12	2. 9.0	3. 1.3333333	4. 1.0	5. 0	6. 30.0
7. 24.0	8. 1	9. 2.0	10. 6.4	11. 0.0	
12. -1	13. 256	14. 2	15. 3.0		

Ans 5.

1.. $w = \left(\frac{\frac{x}{y}}{z} t \right)^3 + 1 + 1.674 \times 10^{-24} c$

2. $q = 1012 p^{\frac{1}{2}} \left(1 - \frac{p}{100} \right)$

3 $k = \frac{ab}{c} - 2$

Ans 6.

1. Invalid	2. Invalid	3. Valid	4. Invalid	5. Invalid
6. Valid	7. Valid	8. Invalid	9. Invalid	
10. Valid	11. Invalid			

Ans 7.

1. FALSE	2. FALSE	3. FALSE	4. FALSE	5. FALSE
6. FALSE	7. FALSE			

Ans 8.

A ** B ** (2 + B - C) / (D + A) * B / (C * D)

Ans 9.

2 1 5 7 8 9 3 4 6

Ans 10.

```
INTEGER N, M, J, K
READ*, N
M = N / 100
N = N - M * 100
J = N / 10
K = N - J * 10
PRINT*, 'THE HUNDREDS DIGIT = ', M
PRINT*, 'THE TENTH DIGIT    = ', J
PRINT*, 'THE ONES DIGIT  = ', K
END
```

Ans 11.

```
      REAL R, PI, SAREA, VOLUME
      READ*, R
      PI = 3.14159
      SAREA = 4 * PI * R ** 2
      VOLUME = 4.0 / 3.0 * PI * R ** 3
      PRINT*,'RADIUS = ', R
      PRINT*, 'AREA = ', SAREA
      PRINT*, 'VOLUME = ', VOLUME
      END
```

Ans 12.

```
2 * X + Y / 2
(( A + B ) / ( A - B )) ** 0.5
R ** 3 / 3.0 - A * C ** ( 3.0 / 4.0 ) / ( 2 * B )
1 / ( 1 / R1 + 1 / R2 + 1 / R3 )
B + X * Y / ( C + D ) + 2
2 * A + C ** (-6)
( A + B ** ( 1.0 / 4.0 ) ) / ( 2 / (A **2 + 5 ) ) - 1
```

Ans 13.

```
A * B * C / ( X * Y ) ** 2
(( A + B ) ** 2  + ( 3 * C )** 3) ** ( A / B )
( A - B + C ) + D * E
( C * X ) ** ( ( 2 - A ) * B )
-B + ( B ** 2 - 4 * A * C ) ** 0.05
```

Ans 14.

```
      INTEGER  SECNDS , MINTS , HOURS , QUAN
      READ*, QUAN
      HOURS  = QUAN / 3600
      QUAN   = QUAN - HOURS * 3600
      MINTS  = QUAN / 60
      SECNDS = QUAN - MINTS * 60
      PRINT*, HOURS,'HOURS',MINTS,'MINUTES',SECNDS,'SECONDS'
      END
```

Ans 15.

```
      INTEGER K1, K2
      READ*, K1 , K2
      PRINT*, K1 , K2
      READ*, K1 , K1 , K2
      PRINT*, K1 , K2
      READ*, K1 , K1 , K1 , K2
      PRINT*, K1 , K2
      END
```

Ans 16.

i) 0

ii)

```
      READ*, K1
      READ*, K2
      READ*, K3 , K4 , K5
```

Ans 17.

1. F2. T 3. T 4. T 5. T 6. T
7. F8. T 9. F 10. F 11. F