
UNIT 3 CONCEPT IN PROGRAMMING LANGUAGE

| Structure | Page No. |
|---|----------|
| 3.0 Introduction | 58 |
| 3.1 Objectives | 58 |
| 3.2 Problem, Algorithm, Program and Programming Language | 59 |
| 3.3 Concept of a Programming Language | 65 |
| 3.4 Elements of Programming Language | 67 |
| 3.4.1 Variables, Constants, Data Type, Array and Expression | 67 |
| 3.4.2 Conditional and Looping Statement | 72 |
| 3.4.3 Subroutine and Function | 73 |
| 3.5 Editor, Assembler, Interpreter & Compiler | 76 |
| 3.6 Summary | 78 |
| 3.7 Answers to Check Your Progress | 80 |
| 3.8 Further Readings | |

3.0 INTRODUCTION

A Programming Language is used to design and describe a set of instructions and computations to be executed by a computer. To do programming, one should have knowledge of i) a particular programming language ii) algorithm to solve the problem. An algorithm is finite number of steps, which perform some computation on the input and produce the desired output in a finite amount of time. Once an algorithm is chosen, the next step for a solution using a computer would be to write a program using the algorithm. A program is defined as a collection of Statements or Sentences written in a particular programming language. Further, to obtain the output from a Program, it is required to be compiled or interpreted so that the computer can understand it.

3.1 OBJECTIVES

After going through this unit, you will be able to :

- need for Programming;
- flow chart and Example Program;
- elements of Programming Languages such as variable, constant, data type, arrays and expression etc.;
- describe looping and decisions; and
- differentiate between Assembler, Compiler and Interpreter.

3.2 PROBLEM, ALGORITHM, PROGRAM AND PROGRAMMING LANGUAGE

A **Problem** is to carry out a particular task. For solving the problem, some input has to be given to the system. The system will process or manipulate the input and produce the desired output. An algorithm describes the steps required to solve a problem. Once an algorithm is obtained for solving a problem, a Program has to be written which a computer can execute so as to accomplish the given task, thereby solving the problem at hand. The program can be in any suitable programming language and is not dependent on the algorithm in any way.

Algorithm: Once a problem has been defined precisely, a procedure or process must be designed to produce the required output from the given input. Since a computer is a machine that does not possess problem-solving judgmental capabilities, this procedure must be designed as a sequence of simple and unambiguous steps. Such a procedure is known as an algorithm.

The steps that comprise an algorithm must be organized in a logical, clear manner so that the program that implements this algorithm is similarly well structured. Number of steps in the algorithm should be finite, they should be executed in finite amount of time and they should give the desired output. Algorithms are designed using three basic methods of control:

- a) **Sequential** : Steps are performed in a strictly sequential manner, each step being executed exactly once.
- b) **Decision/Selection** : One of several alternative actions is selected and executed.
- c) **Repetition** : One or more steps is performed repeatedly.

Any algorithm can be constructed using basic methods of control.

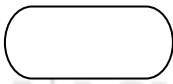
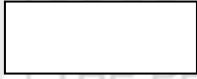
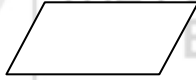
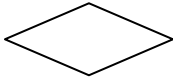
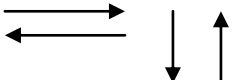
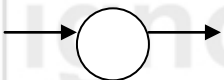
Programs to implement algorithms on the computer must be written in a language that the computer can understand. It is fruitful, therefore, to describe algorithms in a language that resembles the language used to write computer programs. This is called pseudo code. It is not a programming language with a rigid syntax, but is similar to one. The idea is that it should be easy to write a program by looking at the pseudo code..

Let us take few problems to illustrate how to express the solution to a problem in the form of an algorithm. We will also see how algorithm can be diagrammatically represented using flow chart, and also how a program can be written based on the algorithm.

For providing a solution to any problem some input from the user is required. In the following examples the number n that is expected from the user should be an

integer. In the program, it will be declared of the required type only, referred as data type of the variable. A Detailed description of variable and data type is given in a later section.

Flowcharting: A Flowchart is a graphical representation of an algorithm. It can be compared to the blueprint of a building. Just as a building contractor refers to a blueprint for the construction of a building, similarly a programmer refers to a flowchart for writing the program which describes what operations are to be carried out and in what sequence to solve a problem. Flowcharts are usually drawn using some standard symbols. The Basic flowchart symbols are as below:

| | | |
|------------------------------|----------------|---|
| Terminal | Start, End |  |
| Computational processing | or Process |  |
| Input/Output Operation | Input-Output |  |
| Decision making or Branching | Decision |  |
| Flow Lines | Flow Direction |  |
| Joining of two parts | Connector |  |

The number n is expected to be an integer.

Example 1

Problem statement: To find out whether a given number is even or odd.

Algorithm:

Step 1 Start

Step 2 INPUT the number n

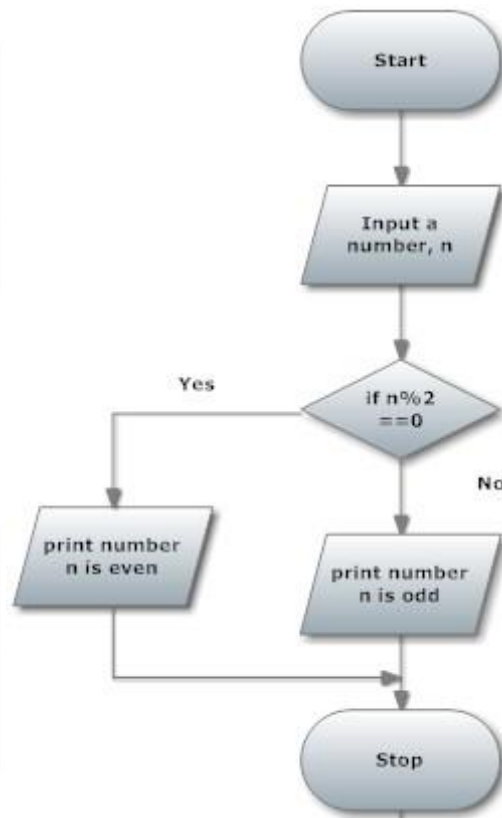
Step 3 Find the remainder on dividing the given number by 2

if (remainder = 0) then
print “number is even”
else

print “number is odd”

Step 4 End

Representing this algorithm through flowchart helps in easier and better understanding of the algorithm :



The program to implement this so that it can be run on a computer can be written in any of the known programming languages. For example, here C has been used as the Programming language for writing the program:

```

#include<stdio.h> /* including header file that has definition of inbuilt functions
*/
void main()
{ /* To mark beginning of block */
    int n; /* variable declaration */

    printf("Enter the number"); /* predefined output function in header file to
display the message on standard output device */
    scanf("%d",&n); /* predefined input function for taking an input from the user
*/
    if (n %2 ==0) /* if else condition to check a expression is true or false and
branch accordingly as per syntax of C programming */
    {
        printf("Number %d is even",n);
    }
    else
    {
        printf("Number %d is odd",n)
    }
} /* to mark the end of block */
  
```

Example 2

Problem: To find the product of first n natural numbers.

Step 1 Start

Step 2 Input number n

Step 3 Initialize product=1

Step 4 Initialize the counter, i=1

Step 5 Compute product=product * i

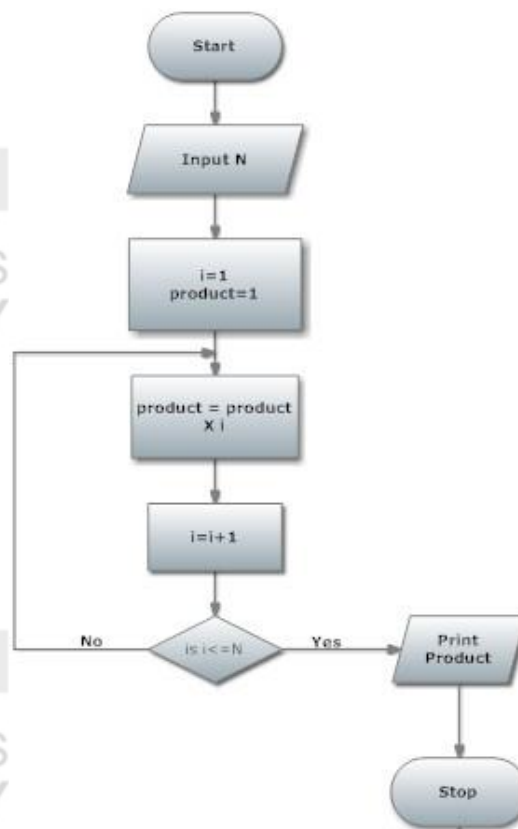
Step 6 Increment counter i by 1 i.e i=i+1

Step 7 Check counter <= n if yes go to step 5

Step 8 Print Product of first n natural numbers as product

Step 9 End

We now express this algorithm as a flowchart for better understanding of the **algorithm**



Here is the C program corresponding to the above algorithm:

```
#include<stdio.h>
```

```
void main()
```

```

{
    int n,i;
    int prod=1;
    printf("Enter number n :");
    scanf("%d",&n);
    for(i=1;i<=n;i++) /* For loop construct for repeating a set of statement n
number of times */
    {
        prod=prod * i;
    }

    printf("Product of first %d natural numbers is = %d",n,prod);
}

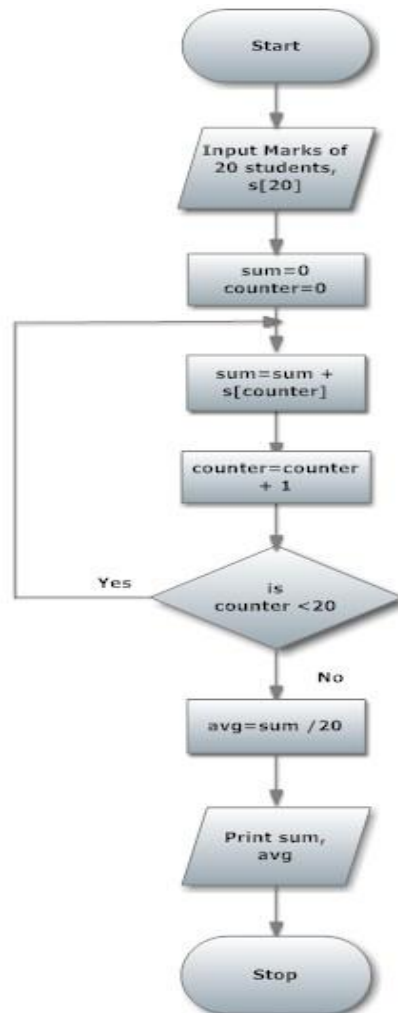
```

Example 3

Problem: To find the sum and average of marks obtained by 20 students in some subject of a course.

Algorithm:

- Step 1 Start
- Step 2 Initialize an array s for 20 students marks i.e s[20]
- Step 3 initialize sum=0
- Step 4 initialize counter=0
- Step 5 Compute sum=sum+s[counter]
- Step 6 increment counter =counter+1
- Step 7 check counter <20
- Step 8 if yes goes to step 5
- Step 9 avg=(sum/20)
- Step 10 Print sum and average
- Step 10 End



Here is the corresponding C program:

```
#include <stdio.h>

void main()
{
    int i, sum; /* declaring variables */
    int s[20]; /* declaring array to refer set of memory locations of same data
type with one name */
    float avg;

    sum=0; /* assignment statement */

    printf("Enter marks for 20 students");
    for(i=0;i<20;i++)
    { printf("%d =",i+1);
      scanf("%d",&s[i]);
    }
    i=0;
    while(i<20) /* while loop construct to repeat set of statement till the condition
is true */
    {
        sum=sum+s[i]; /* arithmetic statement for addition */
```

```

    i++; /* increment statement */
}
avg=sum/20;
printf("Sum of marks for 20 students:=%d",sum);
printf("Average marks of 20 students:=%0.2f",avg);
}

```

Check Your Progress 1

1) Write an algorithm for solving the following problems:

a) To calculate the area of a rectangle.

.....

b) To find the sum of the first n natural numbers

.....

2) Draw a flowchart for a) and b) in Question 1

.....

3.3 CONCEPT OF PROGRAMMING LANGUAGE

In order to communicate with other human beings, one needs some language or a medium. In the same way for solving problems, a programming language is required in order to communicate with the computer. A Programming Language is used to design and describe a set of instructions/actions for the computation that we wish to be executed by the computer.

First an algorithm is designed which solves the problem in question. In the next step, the algorithm is expressed in the form of a program. For this one should have a sound knowledge of some programming language. A program written in a particular Programming Language can control the behavior of a computer and perform a designated task to generate the desired output for any given input.

Program and Programming Language : A Program is defined as a collection of statements/ instructions that enable the computer to solve a problem. The process of writing a program is called programming. Statements in a program should be well formed sentences. Each Programming Language has some basic building blocks called the primitive building blocks. These blocks are defined by basic components that have an associate Syntax and Semantics. Syntax provides structure and semantic provides meaning to it. Different computer programming languages have different advantages and disadvantages to their use. Hence different application domains based on their requirement and functionality may choose any one of the available programming languages suitable for the task.

Syntax provides the structure and how to formulate the phrase or sentence w.r.t grammar of the language. It tells us about which composition is allowed from the character set of the language. Each sentence must be a well formed sentence according to the grammar of the language. The grammar is expressed in a Number of rules that will be finite and these allow the formulation of any number of sentences. A language is defined in the form a quadruplet $L(T,N,P,S)$ where T is set of terminals, N is a set of non terminals, P is set of productions or rules and S is the start symbol. For any language we must have an alphabet/character set, operators and rules. To form a grammar w.r.t a language rules need to be formed. The basic structure of rule is LHS and RHS with some set of terminal and non terminal symbol.

Syntax comprises grammar and vocabulary whereas syntactic analysis is known as parsing. Semantics provides the meaning for the formulated /composed phrase or word or sentence. Semantic function can be incorporated inside the logic of a compiler or interpreter which evaluates ordering of statements for execution.

$S \rightarrow A$

$A \rightarrow Ab \mid b$ i.e any word with sequence of any number of occurrence of character b

Start symbol:

Here S is start symbol. Any sentence will start with start symbol only.

In respect of BNF notation it is as follows:

$L(T,N,P,S)$

$T = \{b\}$

$N = \{A\}$

$P = \{S \rightarrow A, A \rightarrow Ab\}$

S=Start symbol

Example grammar:

Program: statement

Statement: stmt | stmt

Stmt: var=expression

Var: a|b

Expression: term +term | term-term

Term: var/const

e.g $x=y+2$

Similarly

Sentence: subject,verb,object

Subject: noun/article

e.g Ram ate biscuits.

Here each symbol on the left is described in terms of its components. Thus a program consists of statements, which are of the form of an assignment of a variable to an expression, and so on.

Any number of sentences can be formed with the help of a grammar defined for a language. The grammar should be unambiguous. otherwise during syntactic analysis at some point of time it can have more than one meaning.

3.4 ELEMENTS OF PROGRAMMING LANGUAGE

Learning a programming language requires understanding of concepts such as representation of different types of data in the computer, the various methods of expressing mathematical and logical relationships among data elements and the methods for controlling the sequence in which operations can be executed for inputting the data, processing it and generating the output data.

3.4.1 Variables, Constants, Data type, Array and Expression

These are the smallest components of a programming language.

For writing a program, one must know how to use the internal memory of a computer. A Computer memory is divided into several locations where each location has its own address. Memory organization can be represented diagrammatically as below:

| |
|-------|
| Cell1 |
| Cell2 |
| Cell3 |
| Cell4 |
| |
| CellN |

Each location or cell can hold some information. In order to store, retrieve or manipulate data from a specific memory location, we must have some identifier for a particular location.

Variable: As referencing memory by its physical address is very tedious, variable names are used. A variable is a symbolic name given to a memory location. Once a variable is assigned to a memory location, the programmer can refer to that location by variable name instead of its address. Variable is the connection between a name and a value.

It is composed of a name, attribute, reference and a value. Attribute means the type of value a variable can hold.

For example the following programming code in C declares variables a & b.

```
int a,b;  
char c;
```

In the above declaration, a & b are the variable name, which refer to a memory location where integer values can be stored. Instead of address of the memory location, variable names a and b will be used to refer to a memory location in order to store or retrieve update its value.

Similarly, c is a variable name given to a memory location where a character value can be stored. Further c will be used to refer to the memory location in order to store or retrieve or update its value.

Constant : In contrast to a variable, which is used as identifier for a value and which can change, constants are identifiers that are used for values, which cannot be changed. In other words constants are symbols used to refer to quantities which do not change throughout the life of a program. No assignment can be done to a constant.

A numeric constant stands for a number. This number can be an integer, a decimal fraction, or a number in scientific (exponential) notation. Some of the operations which can be performed with numeric constants are addition, subtraction, multiplication, division and comparison.

A string constant consists of a sequence of characters enclosed in double/single quote marks. Chopping off some of the characters from the beginning or end, adding another string at the end (concatenation), copying are some of the operations that performed on the string constants. All these operations can be done on variables also.

For example in the C programming language

-integer constant is declared as:

```
const int a=2; /* the value of a cannot be changed throughout the program*/
```

-string constant is declared as:

```
char const *str; /* str ,string can not be altered by string library functions*/
```

Data Type: Anything that is processed by a computer is called data. There are different types of data that can be given to the computer for processing. A data type is a classification identifying the type of data. It determines the

- Possible values for that type,
- Operations that can be performed on values of that type,
- The way values of that type can be stored in memory,

In each programming language there are some primitive data types. For example in the C programming language they are: Please note that these sizes can be compiler or machine dependent in the case of this language. For other languages such as Java, the sizes are defined by the language itself.

- **int**, both signed and unsigned integers, 2 bytes in size.
- **float, floating point numbers**, up to 4 bytes in size.
- **double, floating point number with double precision**. These are organized in 8 bytes (64 bits)
- **char, character type** size of 1 byte (8 bits) It is used to form the strings i.e sequence of characters.

Array : In programming, when large amount of related data needs to be processed and each data element is stored with different a variable name, it becomes very difficult to manage and manipulate. To deal with such situations, the concept of array is used.

An array is a set of elements of same data type that can be individually referenced by an index. Usually these are placed in contiguous memory locations. Generally two types of array are used:

- One dimensional array
- Two dimensional array

One dimensional array: A one-dimensional array is a structured collection of elements that can be accessed individually by specifying the position of a component with index/ subscript value. The index would let us refer to the corresponding value. value at that .

Like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

type name [elements];

where type is a valid data type (like int, float...), name is a valid identifier or variable name and the elements field (which is always enclosed in square brackets []), specifies how many of these elements the array will contain. Therefore, in order to declare an array named as marks, that will store marks for 5 students,

```
int marks[5];
```

```
marks [0] marks[1] marks[2] marks[3] marks[4]
```

| | | | | |
|----|----|----|----|----|
| 50 | 70 | 80 | 90 | 63 |
|----|----|----|----|----|

Two dimensional Arrays : A two-dimensional array is like a table, with a defined number of rows, where each row has a defined number of columns. In some instances we need to have such an array of arrays. It will have two dimensions and data is represented in the form of rows and columns.

Type name [elements] [elements];

For example int a [3] [3]; /* lets one dimension depict location and other dimension represent sales in a day or a week or a month*/

| | Column1 | Column 2 | Column 3 |
|------|---------|----------|----------|
| Row1 | a[0][0] | a[0][1] | a[0][2] |
| Row2 | a[1][0] | a[1][1] | a[1][2] |
| Row3 | a[2][0] | a[2][1] | a[2][2] |

Expression : An expression is a combination of variables, constants and operators written according to the syntax of the programming language. In the C programming language every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Every computer language specifies how operators are evaluated in a given expression. An expression may contain

- i) arithmetic operator:
- ii) relational operator
- iii) logical operator

Assignment : It is composed of variable name, an assignment operator of the language and a value or variable or some expression as per composition allowed based on rules defined in grammar.

```
e.g temp=5;
temp=temp+1;
```

This means to add 1 to the current value of the variable temp and make that the new contents of the variable temp

```
temp = a+b ;
```

Arithmetic : These types of expressions consist of operators, operands or some expression. The following is the list of arithmetic operator.

- +(addition),
- (subtraction),

*(Multiplication),
/(Division),
% (modulo),
++(increment by 1),
--(decrement by 1)

Here are some examples of arithmetic expressions.

e.g. $x=y+z$; /* addition of y and z will be stored in x */
 $i++$; /* i will be incremented by 1 i.e $i=i+1$ */
 $y=x\%2$; /* remainder after x divided by 2 will be stored in y */

Logical, relational and equality : these types of expression result in a Boolean representation i.e. each expression will result in either True or False. It is composed of operands and relational/logical/equality operator.

The following is the list of operators in the C programming language

== (equal to)
 != (Not equal to)
 < (less than)
 <= (less than equal to)
 > (greater than)
 >=(greater than equal to)
 && (logical AND)
 || (logical OR)
 ! (logical NOT)

Relational expressions result in one of the truth value either TRUE or FALSE. They are capable of comparing only two values separated by any valid relational operator.

e.g.

Let $x=1$, $y=3$
 $x==1$ /* evaluates to true as x has value 1 */
 $x!=y$ /* evaluates to false */
 $x<y$ /* evaluates to true */
 $(x<2) \&\& (y> 5)$ /* evaluates to true */

Bit Wise: Bitwise operators modify variables considering the bit patterns that represent the values they store.

& AND (Binary operator)
 | inclusive OR (OR)
 ^ exclusive OR (XOR)
 << shift left.
 >> shift right.
 ~ one's complement

e.g. let a=2 (0000 0010),b=5(0000 0101)
c=a&b; (0000 0000) /* c=0*/

3.4.2 Conditional and Looping Statement

Conditional statement: An If statement is composed of three parts. The first part should be keyword w.r.t language to convey to the computer that it is if statement. And a Boolean expression. The second and thirds part can be a statement or group of statements as defined in rules of grammar of language.

Generally, an if statement is evaluated and executed in the following sequence: first it evaluates the boolean expression. If the expression is true, the statements in the second part are executed. Otherwise if it is *false*, the statements in the third part are executed. The third part is optional; if it is absent and the expression is false, then the program control simply moves on to the next statement in the sequence.

For example,

```
if (n %2 ==0)
{
    printf("Number %d is even",n);
}
else
{
    printf("Number %d is odd",n)
}
```

Looping Statement: The purpose of a loop structure is to repeat certain tasks until some condition is satisfied. Several variations of a loop structure are available in each programming language to handle different situations.

A program loop consists of two segments, one is the body of the loop and the other is the control statement. The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop. The test may be either to determine whether the loop has repeated the specified number of times or to determine whether the particular condition has been met.

Thus a loop consists of :

- Initial condition
- Execution of set of statements inside the loop
- Test the condition
- Again execute the statements if the condition is met else go to the next statement in the sequence

There are three variants of looping statements in the C programming language are:

- For loop
- While loop
- Do while loop

In this brief introductory unit, we will not go into the details of the distinctions between these three types of loops.

e.g 1

```
for(i=0;i<20;i++)
```

```
{ printf(“%d =”,i+1);
  scanf(“%d”,&s[i]);
}
```

e.g 2

```
i=0;
while(i<20)
{
    sum=sum+s[i];
    i++; /* increment counter */
}
```

Basic structure or keywords may vary in different languages. Also loop structure may be structured or not as it might not have control variables. Most of the languages do have control variables in their loop structure.

3.4.3 Subroutine and Functions

In a program, it is often necessary to repeat a statement or group of statements at several points to accomplish a particular task. Repeating the same statement in a program each time makes a program lengthy and reduces readability. These problems could be sorted out if the necessary statements could be written once and then referred to each time they are needed. This is the purpose of a subprogram. Basically there are two different types of subprograms, called functions and subroutines.

Making subprograms allows tackling small pieces of a problem individually. Once each piece is working correctly then the pieces are integrated together to create the complete solution of the problem. To implement functions and subroutines, we require writing the main program that references all of the subprograms in the desired order and also writing the subprograms. This can be done in any order that is convenient.

The following steps take place during the execution of subprograms:

- 1) Temporarily halt the execution of the calling program i.e main program.
- 2) Execute subprogram.
- 3) Resume execution of the calling program at the point immediately following the call of the subprogram.

Subroutine : A subroutine is a type of subprogram, a piece of code within a larger program that performs a specific task and is relatively independent of the remaining code.

It is also called a procedure, routine or a method.

A subroutine has no value associated with its name. All outputs are defined in terms of arguments; there may be any number of outputs.

In most cases, a subroutine needs some information about the circumstances in which it has been called. A procedure that performs repeated or shared tasks uses different information for each call. This information consists of variables, constants, and expressions that you pass to the procedure when you call it.

A parameter represents a value that the procedure expects you to supply when you call it. You can create a procedure with no parameters, one parameter, or more than one. The part of the procedure definition that specifies the parameters is called the parameter list.

An argument represents the value you supply to a procedure parameter when you call the procedure. The calling code supplies the arguments when it calls the procedure. The part of the procedure call that specifies the arguments is called the argument list. For example here is a subroutine to find the sum of three numbers

```
SUBROUTINE sub1 (A,B,C, SUM)
    REAL A,B,C, SUM
    SUM = A + B + C
    RETURN
END
```

The subroutine sub1 in the main program will be invoked as follows
CALL sub1(A,B,C, SUM)

Function : The purpose of a function is to take in a number of values or arguments, do some calculations with those arguments and then return a single result.

Each language has different rules to define a function. In the C programming language the basic block for function is given as:

return value function name (argument list)

```
{
statement;
}
```

Functions can be called from the main program or from anywhere else, even from within itself. Program control will transfer to function definition statement as soon they are called and then return back to next statement immediately after the calling point.

e.g

```
#include<stdio.h>

void main()
{
int x, y;
printf("Enter number");
scanf("%d",&y);
x=funname(y);
if(x==1)
printf("Number %d is even",y);
else
printf("Number %d is odd",y);
}

int funname(int a)
{
if((a%2)==0)
return 1;
else
return 0;
}
```

Library function: These are the functions supplied with the programming language. The code or definition of library functions does not have to be written in a user program while writing it. Coding or definition of these function are defined in header or library files which are required to be included in program.

e.g.

```
#include<stdio.h>
```

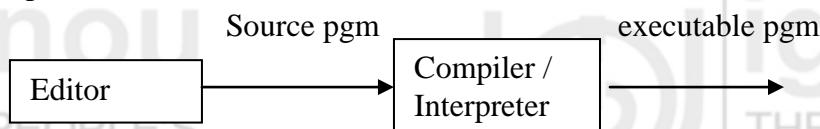
printf(),scanf() etc. are functions defined in stdio.h header file.

Similarly every programming language has a set of library or header files.

3.5 EDITOR, ASSEMBLER, INTERPRETOR & COMPILER

To write a program in any of the programming languages requires an editor. This is a program that is used to create text files. While saving the program, filename and extension as per programming language is required to be given e.g in C programming language f1.c, in C++ f1.cpp or f1.C, in Java f1.java etc. The extension may also depend on the conventions of the operating system used, for instance, in unix the extension for a C++ program is .C while for Windows it would be .cpp.

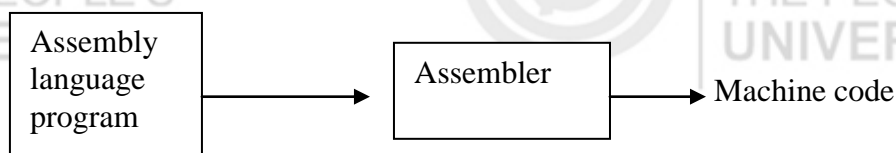
There are different types of editors. Some of the programming languages have some specific built in editors.



A Programming Language is different from machine language, which is understood by a computer in the sense that it can be directly executed. Hence a program in any higher level programming language like C requires a translation process that can translate the source program into machine code so that it can be executed by the computer.

As you may already know from a previous unit, programming languages can be low level languages or high level languages.

Assembly language is a low level programming language similar to machine language, but far easier to write and understand because machine language binary instructions and operands are replaced by mnemonics that are comprehensible to humans. Just As a program written in programming language requires a translator to translate the source program in machine code, a program written in assembly language uses the utility named as assembler for translation purpose. Assembly language is the most basic programming language available for any processor. With assembly language, a programmer works only with operations implemented directly on the physical CPU. Assembly language lacks high-level conveniences such as variables and functions, and it is not portable between various families of processors.



High level programming languages provide:

- Good readability
- Portability

- Easy debugging
- Easy software development

Hence Programming languages translators are broadly divided into two categories:

- Compilers
- Interpreters

Compiled Language : An additional program called a compiler translates a program written in a programming language; into a new file that does not require any other program to execute itself, such a file is called an executable.

e.g. C, C++, Pascal are languages that are typically compiled

Compilers produce better optimized code that generally runs faster and compiled code is self-sufficient and can be run on their intended platforms without the compiler present.

Interpreter : An interpreter is a program that translates each statement in the programming language into machine code and runs it. Such an arrangement means that to run the program one must always have the interpreter available.

e.g Basic , Prolog, Perl are languages that are typically interpreted.

Programs in any language can be interpreted or compiled. So there are basic compilers available as well. Compiled code runs faster and does not need the compiler at run time, whereas interpreted code is slower and needs the interpreter every time the program has to be run.

Check Your Progress 2

1) What is the need of programming language?

.....
.....
.....

2) What is the purpose of looping statements in a programming language?

.....
.....
.....

3) What are basic operators in any of the programming language?

.....
.....

4) What is the purpose of using an array in a programming language?

3.6 SUMMARY

This unit helps in understanding the requirement of programming languages. Broad classification of programming languages is discussed. Any programming language allows one to translate an algorithm to a computer program for solving the given problem.

Basic constructs of programming languages are described which include description of syntax and semantics. Various control structures for any programming language like conditional statements, arrays, loops and subroutine are discussed. To execute a program in programming language, firstly we should have a program then it should be in the form that is understood by computer. Hence editor is required to write a program. Then program requires compiler or interpreter to translate it into machine code so that it can be understood by computer.

3.7 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

1. a) Algorithm to calculate area of a rectangle

Step 1: Read length of the rectangle.

Step 2: Read breadth of the rectangle.

Step 3: Calculate Area = length X breadth

Step 4: Print area.

Step 5: END

1. b) Algorithm to find sum of the first n numbers

Step 1: read the number (n)

Step 2: initialize fact=1,i=1

Step 3: repeat 4,5 until $i \leq n$

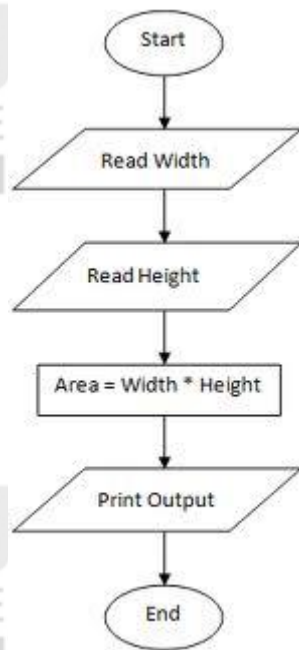
Step 4: fact=fact+i

Step 5: increment i by 1 i.e $i=i+1$

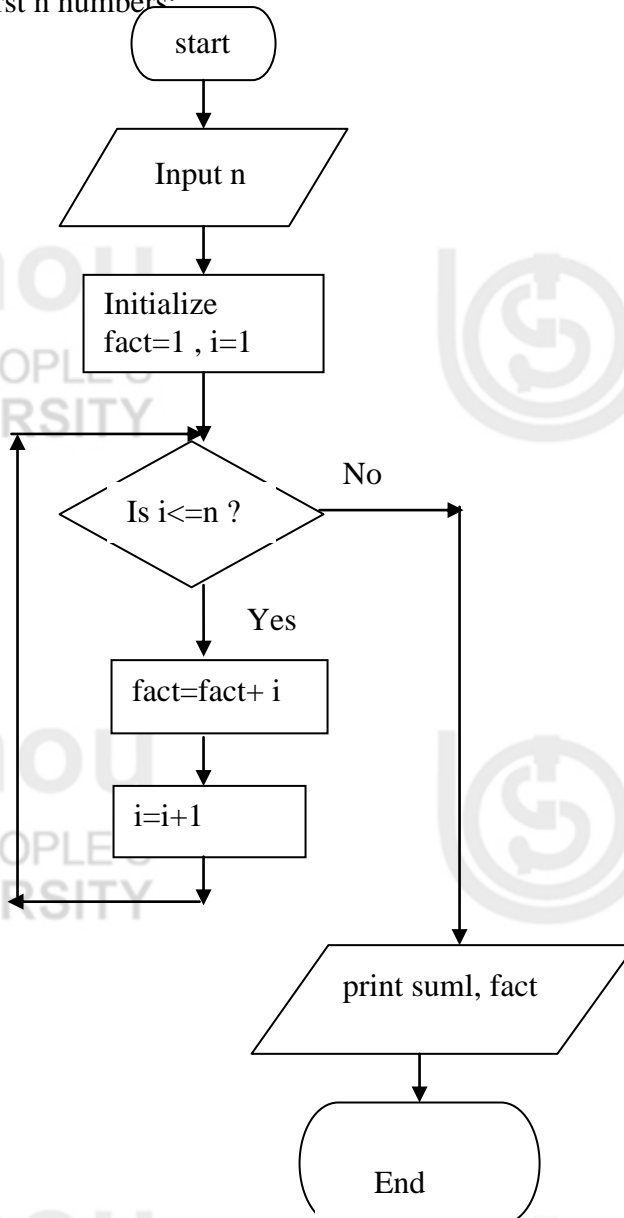
Step 10: Print "Sum of n given numbers is", n , fact

Step 11: END

2. (a) Flowchart to compute area of a rectangle:



(b) Flowchart to find sum of first n numbers:



Check Your Progress 2

1. To solve any problem computer has to be given instructions. Instructions cannot be given in any natural language, which we use (like English, Hindi etc). So it is required to have a programming language to write the program to solve the given problem with the help of a computer.
2. If in a program, a set of statements has to be executed more than once at a particular place, then looping statements are used.
3. The operators in any programming language are broadly classified into the following types:
 - a) **Arithmetic operators:** Operators, which are used to perform arithmetic operations such as addition, subtraction, multiplication, division etc.
 - b) **Logical Operators:** Operators under this category are AND, OR, NOT.
 - c) **Relational Operators :** $>$, $<$, $=$, \neq , $<=$, $>=$ these are the relational operators.
4. In programming, when large amount of related data need to be processed and each data element is stored with different variable name, it becomes very difficult to manage and manipulate. To deal with such situations, the concept of array is used. Array provides a simple & efficient method to refer, retrieve & manipulate a collection of similar data by a single name.

3.8 FURTHER READINGS

Terrence W. Pratt , Marvin V. Zelkowitz “Programming Languages Design and Implementation”, Fourth Edition, 2002, PHI.

Ellis Horowitz “Fundamentals of Programming Languages”, Second Edition, 2000 Galgotia.

R.G Dromey “How to solve by computer”, PHI.