



CONCEPT OF PROGRAMMING AND PROGRAMMING LANGUAGES

CHAPTER - 6

OBJECTIVES OF THIS CHAPTER

- 6.1 Introduction
- 6.2 Concept of Program and Programming
- 6.3 Programming Languages
- 6.4 Language Translator
- 6.5 Programming Process

6.1 INTRODUCTION

In this chapter, we are going to learn about the concept of program, programming, programming process and different categories of programming languages used for computer systems. A program is basically a set of instructions to be executed by computer to perform some task. The process of writing a program is called programming. The person who writes the program is called programmer. When a programmer writes a program, he or she goes through a particular process. This process of developing a program is called programming process. The programmer can use any language from hundreds of available programming languages for program development.

6.2 CONCEPT OF PROGRAM AND PROGRAMMING

We know that a computer system basically consists of two parts: hardware and software. Without software, computer-hardware cannot do anything hence computer is nothing but a piece of metal without software. To make the computer-hardware to do something, we must install and use software in our computer system. Now the question arises what is software?

Software is a set of computer programs which are designed and developed to perform desired tasks on computer. It is the software which makes a computer capable of data processing, storing and retrieval. Basically, softwares are categorised into two types: system software and application software. **System software** are designed and developed to control the functionality & to operate computer system hardware while the application software are designed and developed to perform specific tasks using computer system. System softwares are more complex as compared to application softwares. The development (programming) of system softwares require more skills as compared to application software development.

Software is usually not a single entity. It is a set (collection) of programs. A programmer must write instructions in a prescribed sequence of the programming language being used for development so that the computer system becomes capable of successfully performing the desired task. Thus, we can say that a **Program** is a set of Instructions that the computer executes.

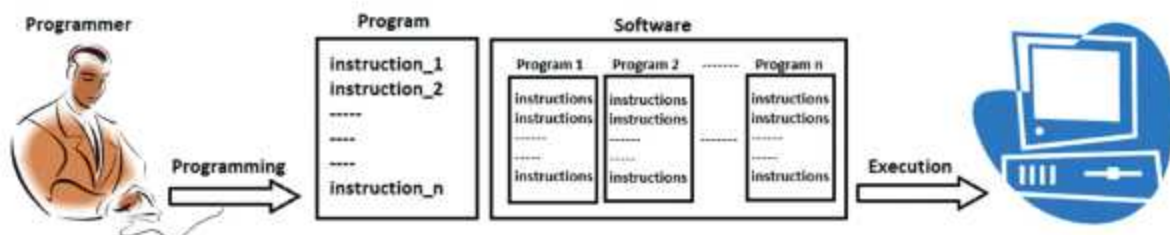


Fig. 6.1 Program, Programmer, Programming and Software

The process of writing system program is known as **System programming** and the programmer for the same is called System Programmer whereas writing application program is known as **application programming** and the programmer for the same is called application programmer.

6.3 PROGRAMMING LANGUAGES

The programming languages are similar to natural languages which are used in our daily life such as Punjabi, Hindi and English etc. As we use natural languages for communication purposes, similarly computer programming languages are used to communicate with the computer systems and to make computers work as desired through softwares.

System programs (Example: Operating Systems) are designed to control & operate the input/output devices, memory, processor etc. To write system program, such as operating system, programmer needs to control the hardware components of computer system. It is possible only if the programmer knows the internal architecture of hardware components. Therefore, system programming is the task of skilled programmers that have a detailed knowledge of the hardware components of the computer system. Machine, Assembly and C languages are widely used to develop system programs.

Application programs are developed to perform a particular task or to solve a particular problem. For example: student management system, library management system, payroll system, inventory control system, word processors, spread sheets, graphics software etc are application programs. Application programmer does not need to possess in-depth hardware knowledge. The most popular application programming languages are PYTHON, COBOL, FORTRAN, BASIC, PASCAL, C, C++, JAVA etc.

6.3.1 Types of Programming Languages

In this era, hundreds of programming languages are available. These languages are categorised on the basis of their ability to develop different kinds of software. Some of the programming languages are best for writing system software while some others are best suitable for writing application software or mobile applications:

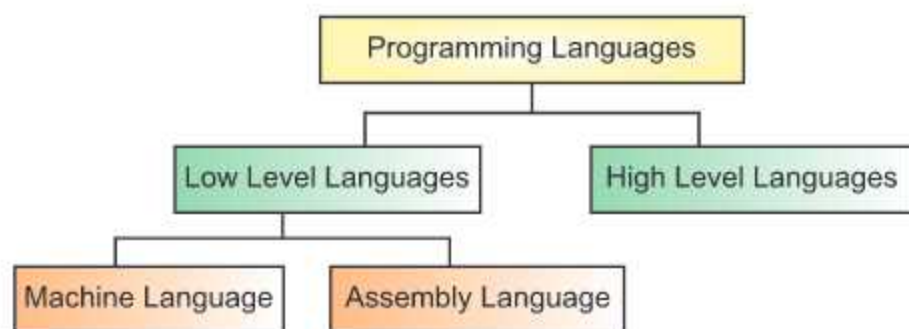


Fig. 6.2 Types of Programming Languages

For designing and developing system programs, low level programming languages are used while for developing application-programs, many high level programming languages have been designed. Let's know more about different type of programming languages:

A. Low Level Languages : Machine language and assembly languages are called low level languages. These programming languages are close to computer hardware and have more direct access to the features of the hardware. These are used to develop drivers, high performance code, kernels for operating systems etc. A detailed explanation of low-level languages is given below:

- a. **Machine Language :** Machine language is also known as **Binary Language**. It is considered as the first generation of computer programming languages. Machine language is the fundamental language for computer systems because this language is directly understood by the computer hardware. Unlike high level programming languages, there is no need for translation of machine language code to make it understandable by the computer. This language consists of only two binary digits - 0 and 1.

Every instruction in machine language consists of two parts: Opcode and operand, as shown in the diagram below:

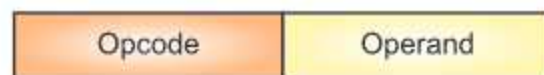


Fig. 6.3 Instruction Format in Machine Language

Here, Opcode is the Operation Code and Operand is the Operation Address. The first part - **Operation code** is the command which tells the computer what operation is going to be performed. The second part - **Operation Address** is the memory address which tells the computer where to find the data for the operation to be performed.

Because the instruction codes are written using binary digits 0 and 1 only, so it becomes difficult to remember the machine instruction codes in binary format. There are many advantages and disadvantages of using machine language some of which are explained below:

Advantages of Machine Language:

- Binary format instructions are **directly understood** by computer without any translation.
- Machine instructions are **executed fast** because these are executed directly without any translation.

Disadvantages of Machine Language:

- It is **difficult to remember** the machine instruction codes as they are made up of complex combinations of binary digits 0 and 1.
- It is the most **difficult process to find errors** in the machine instruction codes.
- Highest level of knowledge of **low-level internal details** of hardware is required for programming in machine language.
- Programs developed in machine language are **machine dependent** because machine instructions are written according to the underlying architecture of computer system. So, these instructions are machine specific which cannot be executed on the computer systems having different architecture.

b. **Assembly Language** : This language is also known as **Symbolic Language** because symbolic names of instructions are used instead of binary codes. This language is considered as second generation of computer programming languages. The major benefit of assembly language as compared to machine language is that it reduces coding time and the amount of information the programmer has to remember. The symbolic names of instructions can be easily remembered therefore it also becomes easy to find errors in the program and to modify it as compared to machine language. Despite of these benefits, programming in assembly language still requires in-depth technical knowledge of hardware. So, programmer must be aware with the machine architecture for programming in assembly language. Due to this hurdle, programs written in assembly language are still machine-dependent. These programs cannot be executed on other machines having different architectures.

Symbolic names used for operation codes in Assembly Language are called **Mnemonic Codes**. For example: the codes for addition, subtraction, multiplication, and division operation are ADD, SUB, MUL and DIV respectively in Assembly language. These codes are the examples of Mnemonic codes.

Now the question arises how do computers execute the assembly language code because computers can execute instructions only in binary format. In order to execute an assembly language code on a computer, it must be translated into equivalent machine understandable code. For this translation, a translator program, named Assembler, is used. Assembler is a language translator program which translates the assembly language code into equivalent machine code. In the following sections, we will study in detail about the assemblers.

Advantages of Assembly Language:

- It is easy to learn and remember the codes of assembly language because it uses English like codes instead of binary digits as compared to machine language.
- Finding and correcting errors in the assembly language program is easy when compared to machine (binary) language.
- Assembly language programs have the equivalent efficiency of the machine language programs.

Disadvantages of Assembly Language:

- Knowledge of low level internal details of hardware is required for programming in assembly language. Therefore hardware technical skills are required for the programmer to do programming in assembly language.
- Programs developed in Assembly language are machine dependent because assembly instructions are written according to the underlying architecture of computer system. So these instructions are machine specific which cannot be executed on the computer systems having different architecture.

B. High Level Languages : The primary objective of developing high level languages is that these languages facilitate a large number of people to build programs or software without the need to know the internal low level details of computer system hardware. These languages are designed to be machine-independent.

High level languages are English like languages. These languages use simple & special characters and numbers for programming. Therefore, these languages make it easy for common people to learn and write computer programs. An instruction written in high level language is usually called a **Statement**. Each high level language has its own rules for writing program instructions. These rules are called Syntax of the language. Some of the commonly used High Level Languages are: PYTHON, BASIC, COBOL, FORTRAN, PASCAL, C, C++, JAVA, C SHARP etc.

Similar to Assembly Language, high level languages cannot be directly understood by computer systems. Language translators are required for translating them into machine understandable format. There are two approaches for translating high level languages into machine code: first is via **Compiler** and other is via **Interpreter**. Each high level language has its own translator program. We cannot translate a program written in one specific high level language with the compiler of some other specific language. For example, we cannot compile C program using COBOL compiler or vice-versa.

Some of the common categories of high level languages are discussed below:

- **Procedural or Procedure Oriented Languages :** Procedural languages are considered as the **Third Generation of Programming Languages (3GLs)**. In procedural languages, a program can be written by dividing it into small procedures or subroutines.

Each procedure contains a series of instructions for performing a specific task. Procedures can be re-used in the program at different places as required. These languages are designed to express the logic of a problem to be solved. The order of program instructions is very important in these languages. Some popular Procedural languages are FORTRAN, COBOL, Pascal, C language etc.

- **Problem-Oriented or Non-Procedural Languages :** Problem oriented languages are also known as Non-Procedural languages. These languages are considered as the **Fourth Generation of Programming Languages (4GL)**. These languages have simple, English-like syntax rules and they are commonly used to access databases. It allows the users to specify what the output should be instead of specifying each step one after another to perform a task. It means there is no need to describe all the details of how the data should be manipulated to produce the result. This is one step ahead from third generation programming languages. These languages provide the user-friendly program development tools to write instructions. Using these languages, user writes the program using application generator that allows data to be entered into the database. The program prompts the user to enter the needed data and then it checks the data for its validity. Examples of problem oriented languages are: SQL (Structure Query Languages), Visual Basic, C# etc. The objectives of these languages are to increase the speed of developing programs and reduce errors while writing programs.
- **Object-Oriented Programming Languages :** The Object-Oriented programming concept was introduced in the late 1960s, but now it has become the most popular approach to develop software. In these programming languages, a problem can be solved by dividing it into a number of objects. Object-Oriented languages support the concept of object, class, encapsulation, data hiding, inheritance and polymorphism etc. Now-a-days, most popular and commonly used Object-Oriented programming (OOPs) languages are C++ and Java.
- **Logic-Oriented languages :** These languages use logic programming paradigms as the design approach for solving various computational problems. Any program written in a logic programming language is a set of sentences in logical form. These sentences express facts and rules about some problem domain. Major logic programming language families include Prolog, Answer Set Programming (ASP) and Datalog. In all of these languages, rules are written in the form of clauses. Such languages are very beneficial in the field of Artificial Intelligence and Robotics.

Advantages of High Level Languages : Some of the common advantages of high level languages are given below:

- High Level languages are easy to learn and understand as compared to low level languages. It is because the programs written in these languages are similar to English-Like statements.

- The errors in a high level language program can be easily detected and removed. All the syntax errors are detected and removed during the compilation process of the program.
- These languages provide a large number of built-in functions that can be used to perform specific task during programming which results in huge time saving i.e. much faster development.
- Programs written in high level language are machine independent. A program written for one type of computer architecture can be executed on another type of computer architecture with little or no changes.

Disadvantages of High Level Languages : Some of the common disadvantages of high level languages are given below:

- A program written in high level languages has lower efficiency & speed as compared to equivalent programs written in low level languages.
- Programs written in high level languages require more time and memory space for execution.
- High level languages are less flexible than low level languages because normally these languages do not have direct interaction with computer's hardware such as CPU, memory and registers.

6.4 LANGUAGE TRANSLATORS

Language translators are also called Language Processors. These are the system programs which are helpful to develop programs. Language translators are designed primarily to perform two main functions as described below:

- These are designed to translate source programs into machine's object code. **Source programs** may be written in Assembly Language or High Level languages while **object code** is a code that a computer CPU can understand without any translation.
- These translator programs are also designed to detect any syntax errors in the source program. Successful translation of source program into object program takes place only if the source program does not have any syntax errors in it.

Each language has its own translator program which can translate the program written only in that specific language. Assembler is a translator program which can translate the source program written in assembly language only. Similarly each high level language has its own translator program, known as Compiler and Interpreter. Some High level languages use Compiler (for example: C/C++ Language) while some other uses Interpreter (for example: BASIC language). But there exists also some languages which have both compiler and interpreter for different levels of translation, for example: JAVA is a language which has both compiler and interpreter. All these types of translator programs are discussed below:

6.4.1 Assembler

It is a language translator which converts assembly language program into machine-understandable format. The program written in assembly language is called Source Program. This source program cannot be directly understood by the computer system. That is why it must be translated into machine understandable format for the execution. It is the assembler which translates this assembly language source program into machine understandable program. The source program after translation (in machine understandable form) is called Object Program (Code). This object program is provided to processor for execution.



Fig. 6.4 Working of Assembler

As shown above the input to assembler is an Assembly Language Program and the output of assembler is a program in the machine understandable form.

6.4.2 Interpreter and Compiler

There are two types of language translators for High level languages i.e. 1. Interpreters 2. Compilers. These translators are used for translating source programs written in High Level languages into machine understandable form.

In the first approach, i.e. Interpreter, one statement of high level language program is taken at a time and it is translated into machine instruction which is executed immediately by the processor. It means no object program is saved in this approach of translation. Whenever we want to execute the program we have to translate the source program every time.



Fig. 6.5 Working of Interpreter

Interpreters do not require large memory space to translate and execute programs. The main disadvantage of interpreters is that they require same amount of time whenever we execute programs on a computer system because every statement of source program must be translated every time.

In the other approach, i.e. Compiler, all statements of the high level language program are taken at a time and they are translated into machine understandable form which is stored as Object Program in Memory. This object program is provided to computer system whenever program is executed. Compilers take more time to translate source program as compared to interpreter. But compiled object program runs much faster than the interpreted program.

Each High Level language has its own compiler. We cannot compile the source code of one language with the compiler of another language. For example FORTRAN compiler cannot compile the source code written in COBOL language and vice-versa.



Fig. 6.6 Working of Compiler

The difference between an interpreter and a compiler may be understood with the help of following analogy. Suppose we want to translate a speech from Tamil to Hindi. We can use two approaches to do this translation. In first approach, translator listens to a sentence in Tamil and immediately translates it into Hindi. In the second approach, the translator listens to the whole passage in Tamil and then gives the equivalent Hindi passage. An interpreter is similar to first approach of translation where sentence-by-sentence translation is carried out whereas compiler is similar to second approach where whole passage is translated in a single step.

6.5 PROGRAMMING PROCESS

We know that a computer needs a program to tell it what to do. Instructions in the program guide the computer how to solve the given problem. But developing a program is not a simple and easy task. A programmer has to go through a specific process for developing a program successfully. The steps involve in the programming process are listed below and are required to be followed in the same sequence:

1. Defining the problem to be solved
2. Plan the solution of the problem
3. Coding the solution in the high level language
4. Compile the program
5. Test and Debug the program
6. Documenting the program

These steps can be explained in detail as follows:

6.5.1 Defining the problem

It is the first step in the programming process. Before a programmer begins his task, he must need to know the extensive details of the problem to be solved through programming. The details of the problem should be provided to the programmer so that he gets a clear understanding of it. Analysis of the problem shows what the required inputs and outputs will be for the solution of problem. After having a clear understanding of the problem, programmer starts thinking about how to solve the given problem.

6.5.2 Planning the Solution

The next step after defining the problem is to prepare a detailed list of steps required to be carried out for solving the problem. We will take an example which shows why planning is required for solving the problem. A teacher asks the student to solve a specific mathematical

problem and the student is not familiar with the steps involved in solving the problem. Thus, he would not be able to solve it. The same principle applies to writing computer programs also. A programmer cannot write the instructions for any program unless he understands how to solve the problem manually.

If a programmer knows the steps for solving the problem but while programming, if he applies the steps in the wrong sequence or he forgets to apply any of these steps, he will get a wrong output or even no output at all. Hence to write an effective program, a programmer has to write all instructions in the correct sequence. Therefore, to ensure that the instructions of the program are appropriate and are in the correct sequence, the programmer must plan the program before start writing it. For planning and defining the steps for the programs, programmers normally use Algorithms and Flow-Charts. Both of these approaches are explained below in detail:

6.5.2.1 Algorithms

The development of an algorithm is basic requirement to computer programming. It is a step-by-step description of how to solve a given problem. An algorithm consists of finite steps and a guaranteed result. When these steps are carried out as specified in the algorithm, it produces the required output. The construction of the algorithm requires creative thinking. Before developing a program, a programmer first set out the algorithm so that he can visualize the possible alternatives to solve the given problem. In order to qualify for the sequence of steps, to be called an algorithm, it should have the following features:

1. Each step should be accurate.
2. Each step should be unambiguous, i.e. it should not have dual meaning.
3. The inputs and outputs should be carefully specified.
4. Steps should not be repeated infinitely.
5. After executing the steps, the required output must be produced.

To gain a clear understanding of algorithms, let us consider some simple examples of defining algorithms.

Problem Statement 1 : Calculate and print multiplication of two numbers:

Algorithm-I:

- Step1: Start
- Step2: Read two numbers A and B.
- Step3: Multiply A and B and store result in C.
- Step4: Print C.
- Step5: Stop.

Problem Statement 2: Write an algorithm to find whether a given number is Odd or Even.

Algorithm-II:

Step1: Start
Step2: Read a Number A
Step3: Divide the number A by 2 and get remainder value
Step4: If remainder of the division is zero then
 Print "Number is Even"
 Else
 Print "Number is Odd"
Step5: Stop

Problem Statement 3: Write an algorithm to print "India is Great" 10 times:

Algorithm-III:

Step1: Start
Step2: Count = 1
Step3: While (Count<=10)
Step4: Print "India is great"
Step5: Count=Count + 1
 [End of While loop]
Step6: Stop

6.5.2.2 Flow Charts

Flow charts are the common ways to represent algorithms. These are also frequently used by the programmers for planning of the program. Programmers often find them very helpful for developing effective and correct programs.


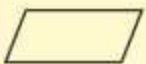


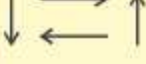

A flowchart is a pictorial representation of the algorithm. Programmers often draw flow charts to visually organize the sequence of steps defined in the algorithm. It uses different types of symbols/shapes to represent different types of instructions. The process of drawing a flowchart for an algorithm is known as flowcharting.

Normally, an algorithm is first represented as flowchart, and the flow chart is then expressed in a programming language to develop a computer program. The main advantage of this two-step approach in program development is that a programmer can more easily detect logical errors in the program logic because a flow chart shows the flow of operations in the pictorial form. Once the flow chart is ready, the programmer can concentrate only on coding of operations. This normally ensures an error-free program.

Experienced programmers, sometimes, write programs without drawing flowcharts. However, beginners should first draw a flowchart to effectively do the programming. Moreover, it is a good practice to have a flow chart along with the computer program. It proves to be very useful during testing of the program as well as during modifications in the program.

Before preparing flowcharts, we have to learn about the different symbols used in the flowcharts. Some of the basic symbols used in the flowcharts are:

Table: 6.1 Symbols used for flowcharts

Symbol	Symbol Name	Name	Description
	Ellipse/Oval	Terminal	It is used to start and terminate the flow chart
	Parallelogram	Input / Output	It is used for taking input data and giving output result
	Rectangle	Processing	It is used for performing computational operations
	Rhombus	Diamond	It is used when we have to choose one path from many paths
	Arrows	Flow Lines	It is used to represent the direction of flow in the flowchart
	Circles	Connectors	It is used to connect different parts of the flowchart.

Now let us consider some examples of flowcharts so that we get familiar with the concept of flowcharts:

a. Draw a flowchart to Calculate and print multiplication of two numbers

(Flowchart for Algorithm-I):

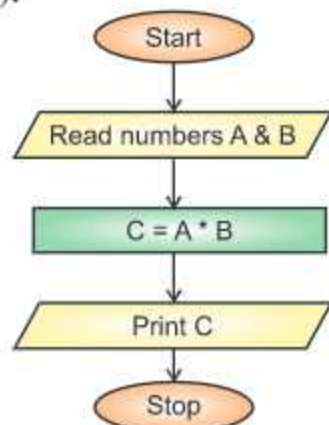


Fig. 6.7 Flowchart for Algorithm-I

b. Draw a flowchart to find whether a given number is odd or even

(Flowchart for Algorithm-II):

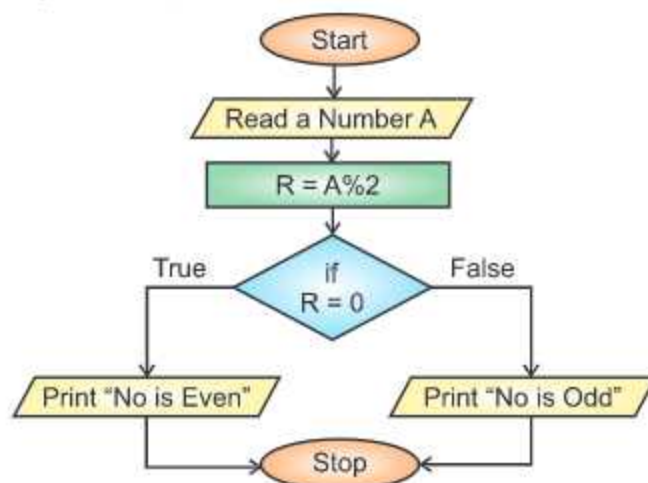


Fig. 6.8 Flowchart for Algorithm-II

c. Draw a flowchart to print "India is great" 10 times

(Flowchart for Algorithm-III):

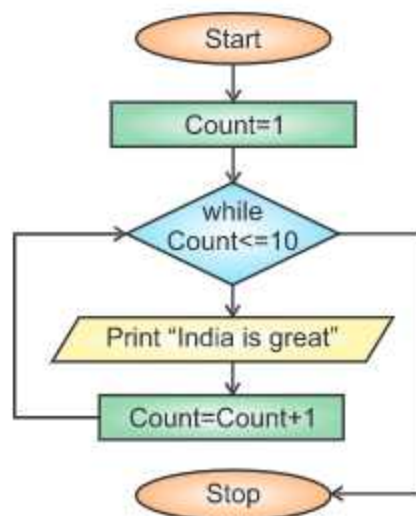


Fig. 6.9 Flowchart for Algorithm-III

Now, we have a clear understanding of how to plan the steps to solve the given problem. Once, inputs, outputs, algorithms and flowcharts have been clearly defined, the next step is to translate these steps into a program using High level programming languages.

6.5.3 Coding the Solution:

The sequence of operations defined in flow-chart can be converted into instruction using some High-Level programming language, such as C, C++, and PASCAL etc. Coding is the process of writing the instructions using programming language to make a program. This program file is known as source program or source code. This code is stored in a disk file. This

program contains the logic or steps for solving the problem. For example, let us consider the following source code using C language for the Algorithm-I:

Source Code for Algorithm-I using High Level Language C:

```
#include<stdio.h>
void main( )           //step1 of the Algorithm-I
{                       //step1 of the Algorithm-I
    int a, b, c;         //step1 of the Algorithm-I
    a=10;                //step2 of the Algorithm-I
    b=20;                //step2 of the Algorithm-I
    c=a * b;             //step3 of the Algorithm-I
    printf("%d", c);     //step4 of the Algorithm-I
}                       //step5 of the Algorithm-I
```

6.5.4 Compile the program

After writing program code in High level language, we have to translate it into such a form that computer can understand & execute it because computer can understand instructions only in binary format. A compiler is a small program that translates the source program into a machine understandable form (i.e. object code). This conversion of source code into object code is known as Compilation. During compilation, compiler also scans the source program for syntax errors. If there are syntax errors in the program, compiler generates error messages. These errors must be corrected to generate the object code. Then the object code will be stored in a disk file. Whenever we want to execute the program, this object code is supplied to computer for execution.

6.5.5 Testing and Debugging the Program

Testing and debugging are important steps in the software development. Testing is a process which makes it sure that program (software) performs the intended task. Testing is a time-consuming task. As long as human beings make programs, the programs will have errors. These program errors are called Bugs. The process of detecting and correcting such bugs is called Debugging. Generally, two types of errors are found in the programs:

- **Syntax Errors :** These errors occur when we do not follow the rules or syntax of programming language being used. These types of errors are automatically detected by compilers during compilation process. A program cannot be successfully compiled until all of the syntax errors in the program are removed. Some examples of syntax errors in C language are: missing semicolon, variable not declared, un-terminated string, compound statement missing etc.
- **Logical Errors :** These errors occur when there are errors in the logic of the program. If our program has logic errors, though it will compile successfully but it may produce wrong result/output. Such types of errors cannot be detected by the compilers.

These errors are either traced out manually by the programmer or some debugging tools may be used to detect such errors. Programmer can detect any faulty logic by examining the output.

6.5.6 Documenting the Program

The final stage in the development process of a program is documentation. The term documentation means specifying the important information regarding the approach and logic applied in the program by the programmer. The documentation enables other programmers to understand the logic and purpose of the program. It is also helpful in the maintenance of the program.



Points To Remember

1. A Program is a set of instructions while a Software is a set of programs.
2. A Programmer is the person who writes the program code.
3. The Process of writing a program is called Programming
4. Machine language is directly understood by computer and consists of binary digits 0 and 1.
5. Assembly Language use mnemonic codes to write instructions in the program.
6. High Level languages use alphanumeric codes to write instructions in the program.
7. Assembler is a language translator which translates Assembly language source program into machine understandable format which is called Object Program Code.
8. Compiler is a language translator which translates High Level language source program into machine understandable format which is called Object code.
9. Algorithm is a set of finite steps to solve some specific problem.
10. Flowchart is the pictorial representation of the algorithm.
11. Writing the instructions to make a program using some computer language is called coding.
12. Finding and correcting errors in the program is called Debugging.

EXERCISE



Part-A

1. Multiple Choice Questions

- I. Set of instructions is called _____.
 - a. Group
 - b. Software
 - c. Program
 - d. None of these

- II. Which language is directly understood by computer without any translation?
- Procedure Oriented Language
 - Machine Language
 - Assembly Language
 - High Level Language
- III. Mnemonic codes & symbolic addresses are used in which programming language?
- Object Oriented Language
 - Non-Procedural Language
 - Assembly Language
 - Machine Language
- IV. Which translator does not save object code after translation of source program written in high level language?
- Translator
 - Compiler
 - Assembler
 - Interpreter
- V. Process of finding and correcting errors in a program is called _____
- Compilation
 - Coding
 - Debugging
 - Documentation

2. Fill in the Blanks:

- I. A person who writes the program is called _____
- II. Low level internal details of hardware are required for programming in _____
- III. _____ is the pictorial representation of algorithm
- IV. Process of translating source program written in high level language into object code is called _____
- V. Those errors which are not detected by the compilers are called _____ errors.

3. Write the Full form of following:

- Opcode
- Operand
- 4GL
- SQL
- OOP

Part-B

4. Short Answer Type Questions. (Write the answers in 4-5 lines)

- What is Programming?
- What are Procedure Oriented Programming Languages?
- Write the names of different symbols used in flowcharts.
- Write the steps used in Programming Process.
- What are Syntax Errors?

Part-C

5. Long Answer Type Questions. (Write the answers in 10-15 lines)

- I. What are low level programming languages? Explain their advantages and disadvantages.
- II. What are Language Translators? Explain any one translator in detail.
- III. What is algorithm? Explain the different features that an algorithm should have.
- IV. Explain different types of errors found in the computer programs.

Lab Activity

- Draw a chart which represents different categories of programming languages.
- Make symbols used in flow charts with the cardboard and label them.

