UNIT 2: PROGRAMMING, ALGORITHMS AND FLOWCHARTS

UNIT STRUCTURE

2.0 OBJECTIVE

After going through this unit, you will be able to

define *computer program*, *algorithm*, and *high-level programming language*.
list the basic stages involved in writing a computer program.
distinguish between high level language and low level language.
describe what compilers and interpreters are and what they do.
apply an appropriate problem-solving method for developing an algorithmic solution to a problem.

2.1 INTRODUCTION

Today, most people don't need to know how a computer works. Most people can simply turn on a computer or a mobile phone and point at some little graphical object on the display, click a button or swipe a finger or two, and the computer does something. An example would be to get weather information from the net and display it. How to interact with a computer program is all the average person needs to know.

But, since you are going to learn how to write computer programs, you need to know a little bit about how a computer works. Your job will be to instruct the computer to do things.

Basically, writing *software* (computer programs) involves describing *processes*, *procedures*; it involves the authoring of ***algorithms***. Computer programming involves developing lists of instructions - the ***source code*** representation of software The stuff that these instructions manipulate are different types of objects, e.g., numbers, words, images, sounds, etc. Creating a computer program can be like composing music, like designing a house, like creating lots of stuff. It has been argued that in its current state it is an *art*, not engineering.

An important reason to consider learning about how to program a computer is that the concepts underlying this will be valuable to you, regardless of whether or not you go on to make a career out of it. One thing that you will learn quickly is that a computer is very dumb, but obedient. It does exactly what you tell it to do, which is not necessarily what you wanted. Programming will help you learn the importance of clarity of expression.

A **programming language** is a formal computer **language** designed to communicate instructions to a machine, particularly a computer. **Programming languages** can be used to create programs to control the behavior of a machine or to express algorithms. The term **programming language** usually refers to high-level **languages**, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal. Each language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

2.2 PROGRAMS AND PROGRAMMING

A computer is a programmable machine. This means it can execute a programmed list of instructions and respond to new instructions that it is given.

Computer programming is a process that leads from an original formulation of a computing problem to executable computer programs. Programming involves activities such as analysis, developing understanding, generating algorithms, verification of requirements of algorithms including their correctness and resources consumption, and implementation of algorithms in a target programming language. Source code is written in one or more programming languages. The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solving a given problem. The process of programming thus often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.

Notice that the key word in the definition of computer is *data*. Computers manipulate data. When you write a program (a plan) for a computer, you specify the properties of the data and the operations that can be applied to it. Those operations are then combined as necessary to solve a problem. Data is information in a form the computer can use—for example, numbers and letters. Information is any knowledge that can be communicated, including abstract ideas and concepts such as “the Earth is round.” Data comes in many different forms: letters, words, integer numbers, real numbers, dates, times, coordinates on a map, and so on. Virtually any kind of information can be represented as data, or as a combination of data and operations on it. Each kind of data in the computer is said to have a specific data type. For example, if we say that two data items are of type **Integer**, we know now they are represented in memory and that we can apply arithmetic operations to them.

Just as a concert program lists the pieces to be performed and the order in which the players perform them, a computer program lists the types of data that are to be used and the sequence of steps the computer performs on them.

Related tasks include testing, debugging, and maintaining the source code, implementation of the build system, and management of derived artifacts such as machine code of computer programs.

Programs are written to solve problems or perform tasks on a computer.

Programmers translate the solutions or tasks into a language the computer can understand. As we write programs, we must keep in mind that the computer will only do what we instruct it to do. Because of this, we must be very careful and thorough with our instructions.

Before getting into computer programming, let us first understand computer programs and what they do. A computer program is a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer. The two important terms that we have used in the above definition are:

• Sequence of instructions

• Computer Programming Language

To understand these terms, consider a situation when someone asks you about how to go to a nearby KFC. What exactly do you do to tell him the way to go to KFC? You will use Human Language to tell the way to go to KFC, something as follows:

First go straight, after half kilometer, take left from the red light and then drive around one kilometer and you will find KFC at the right.

Here, you have used English Language to give several steps to be taken to reach KFC. If they are followed in the following sequence, then you will reach KFC:

1. Go straight
2. Drive half kilometer
3. Take left
4. Drive around one kilometer
5. Search for KFC at your right side

Now, try to map the situation with a computer program. The above sequence of instructions is actually a Human Program written in English Language, which instructs on how to reach KFC from a given starting point. This same sequence could have been given in Spanish, Hindi, Arabic, or any other human language, provided the person seeking direction knows any of these languages.

Now, let's go back and try to understand a computer program, which is a sequence of instructions written in a Computer Language to perform a specified task by the computer. Following is a simple program written in Python programming Language:

```
print "Hello, World!"
```

The above computer program instructs the computer to print "Hello, World!" on the computer screen.

• A computer program is also called a computer software, which can range from two lines to millions of lines of instructions.

• Computer program instructions are also called program source code and computer programming is also called program coding.

• A computer without a computer program is just a dump box; it is programs that make computers active. As we have developed so many languages to communicate among ourselves, computer scientists have developed several computer-programming languages to provide instructions to the computer.

2.3 BUILDING BLOCKS FOR SIMPLE PROGRAMS

While the specific rules of a programming language differ from one language to another, the underlying foundation upon which all programming languages are built is pretty much the same. There is a hierarchy that all languages follow. This hierarchy is:

Programs ⇨ Statements ⇨ Expressions ⇨ Operands and Operators

You can verbalize this hierarchy like this: "Programs consist of one or more statements. Statements consist of one or more expressions. Expressions consist of one or more operands in conjunction with one or more operators."
Every computer requires appropriate instruction set (programs) to perform the required task. The quality of the processing depends upon the given instructions. If the instructions are improper or incorrect, then it is obvious that the result will be superfluous.
Therefore, proper and correct instructions should be provided to the computer so that it can provide the desired output. Hence, a program should be developed in such a way that it ensures proper functionality of the computer. In addition, a program should be written in such a manner that it is easier to understand the underlying logic.
A good computer program should have following characteristics:

- **Portability**: Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Due to rapid development in the hardware and the software, nowadays platform change is a common phenomenon. Hence, if a program is developed for a particular platform, then the life span of the program is severely affected.
- **Readability**: The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort. If a program is written structurally, it helps the

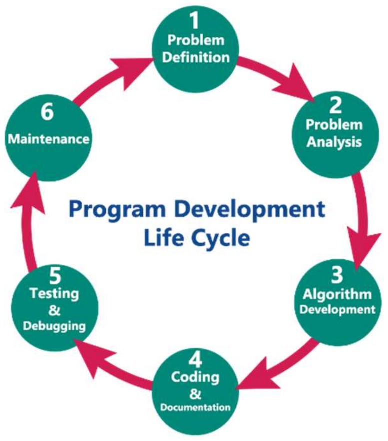
programmers to understand their own program in a better way. Even if some computational efficiency needs to be sacrificed for better readability, it is advisable to use a more user-friendly approach, unless the processing of an application is of utmost importance.

- **Efficiency**: Every program requires certain processing time and memory to process the instructions and data. As the processing power and memory are the most precious resources of a computer, a program should be laid out in such a manner that it utilizes the least amount of memory and processing time.
- **Structural**: To develop a program, the task must be broken down into a number of subtasks. These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask. If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier.
- **Flexibility**: A program should be flexible enough to handle most of the changes without having to rewrite the entire program. Most of the programs are developed for a certain period and they require modifications from time to time. For example, in case of payroll management, as the time progresses, some employees may leave the company while some others may join. Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.
- **Generality**: Apart from flexibility, the program should also be general. Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain. For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.
- **Documentation**: Documentation is one of the most important components of an application development. Even if a program is developed following the best programming practices, it will be rendered useless if the end user is not able to fully utilize the functionality of the application. A well-documented application is also useful for other programmers because even in the absence of the author, they can understand it.

2.4 PROGRAMMING LIFE CYCLE PHASES

When we want to develop a program using any programming language, we follow a sequence of steps. These steps are called phases in program development. The program development life cycle is a set of steps or phases that are used to develop a program in any programming language.
Generally, program development life cycle contains 6 phases, they are as follows….

- Problem Definition
- Problem Analysis
- Algorithm Development
- Coding & Documentation
- Testing & Debugging
- Maintenance



1. Problem Definition
In this phase, we define the problem statement and we decide the boundaries of the problem. In this phase we need to understand the problem statement, what is our requirment, what should be the output of the problem solution. These are defined in this first phase of the program development life cycle.

2. Problem Analysis
In phase 2, we determine the requirements like variables, functions, etc. to solve the problem. That means we gather the required resources to solve the problem defined in the problem definition phase. We also determine the bounds of the solution.

3. Algorithm Development
During this phase, we develop a step by step procedure to solve the problem using the specification given in the previous phase. This phase is very important for program development. That means we write the solution in step by step statements.

4. Coding & Documentation
This phase uses a programming language to write or implement actual programming instructions for the steps defined in the previous phase. In this phase, we construct actual program. That means we write the program to solve the given problem using programming languages like C, C++, Java etc.,

5. Testing & Debugging
During this phase, we check whether the code written in previous step is solving the specified problem or not. That means we test the program whether it is solving the problem for various input data values or not. We also test that whether it is providing the desired output or not.

6. Maintenance
During this phase, the program is actively used by the users. If any enhancements found in this phase, all the phases are to be repeated again to make the enhancements. That means in this phase, the solution (program) is used by the end user. If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

2.5 PSEUDOCODE REPRESENTATION

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading.

Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation. The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm.

The rules of Pseudocode are reasonably straightforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch. Examples below will illustrate this notion.

2.5.1 Examples:

1.

```
If student's grade is greater than or equal to 60
        Print "passed"
else
        Print "failed"
```

2.
```
Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
Set the class average to the total divided by ten
Print the class average.
```

3.
```
Initialize total to zero
Initialize counter to zero
Input the first grade
while the user has not as yet entered the sentinel
    add this grade into the running total
    add one to the grade counter
    input the next grade (possibly the sentinel)
if the counter is not equal to zero
    set the average to the total divided by the counter
    print the average
else
    print 'no grades were entered'
```

4.
```
initialize passes to zero
initialize failures to zero
initialize student to one
while student counter is less than or equal to ten
    input the next exam result
    if the student passed
        add one to passes
    else
        add one to failures
    add one to student counter
print the number of passes
print the number of failures
if eight or more students passed
    print "raise tuition"
```

2.5.2 Some Keywords That Should be Used

For looping and selection, The keywords that are to be used include Do While...EndDo; Do Until...Enddo; Case...EndCase; If...Endif; Call ... with (parameters); Call; Return ....; Return; When; Always use scope terminators for loops and iteration.

As verbs, use the words Generate, Compute, Process, etc. Words such as set, reset, increment, compute, calculate, add, sum, multiply, ... print, display, input, output, edit, test , etc. with careful indentation tend to foster desirable pseudocode.

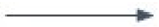
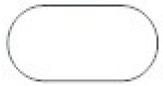
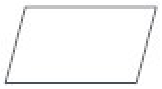
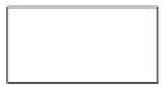
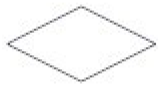
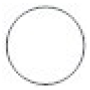
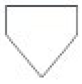
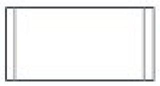
Do not include data declarations in your pseudocode.

2.6 FLOW CHARTS

A **flowchart** is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem.

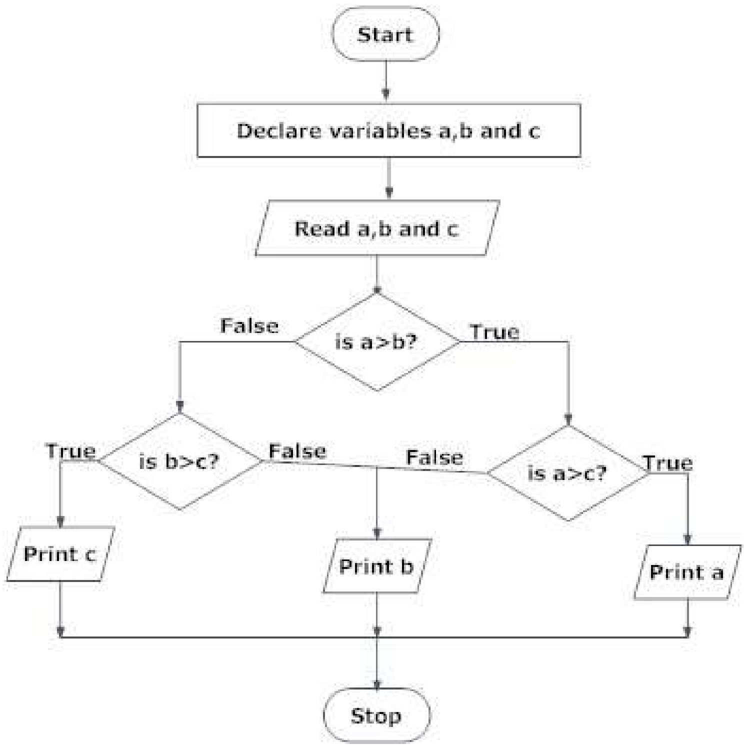
2.6.1 Symbols Used In Flowchart

Different symbols are used for different states in flowchart, For example: Input/Output and decision making has different symbols. The table below describes all the symbols that are used in making flowchart

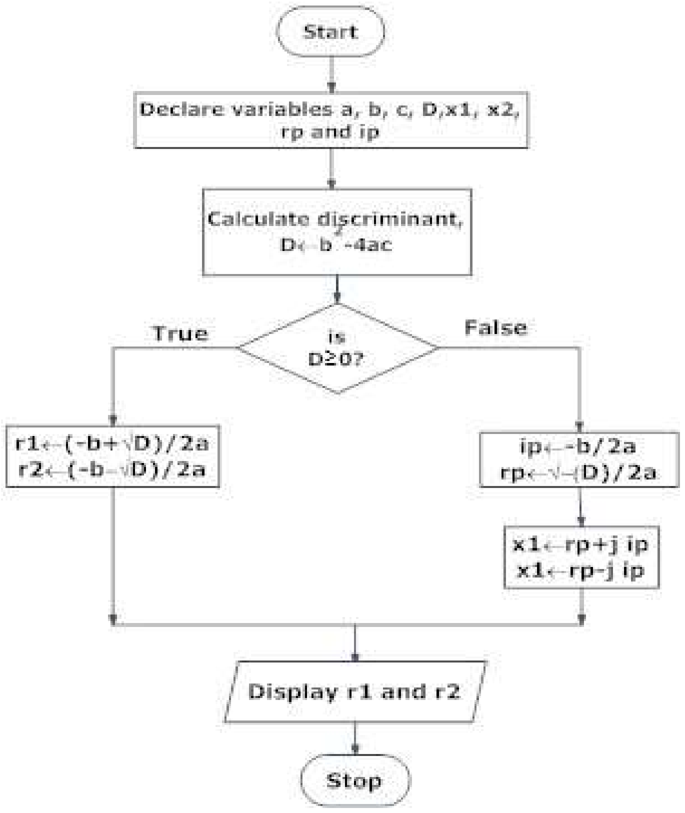
Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.
	Desicion	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/Function	Used to represent a group of statements performing one processing task.

2.6.2 Examples of flowcharts in programming

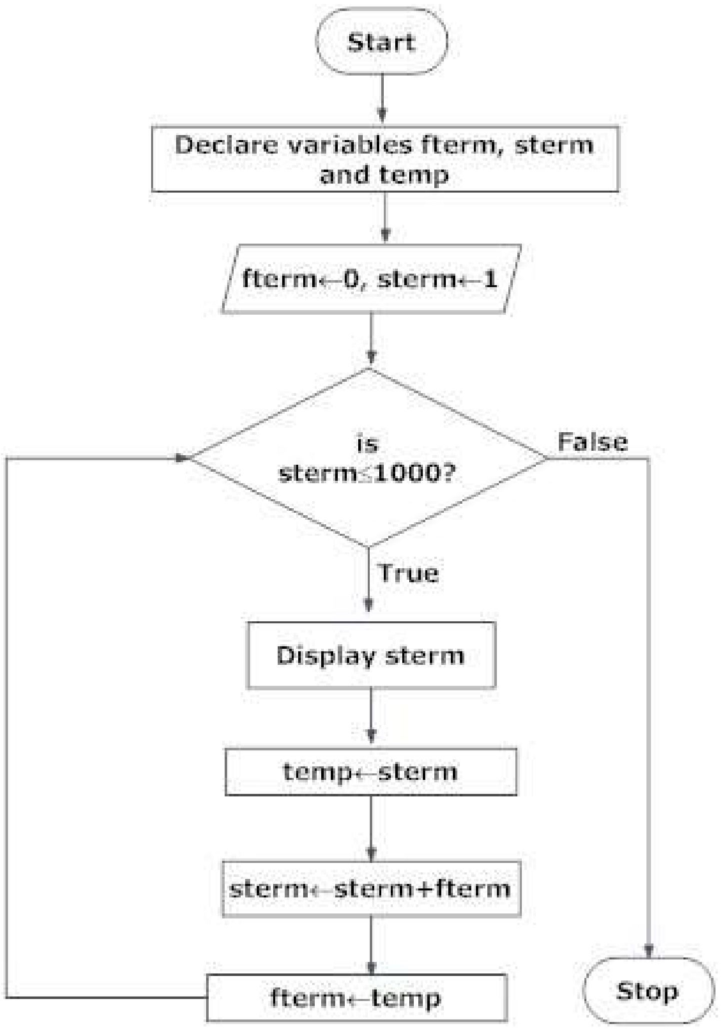
2.6.2.1 Draw flowchart to find the largest among three different numbers entered by user.



2.6.2.2 Draw a flowchart to find all the roots of a quadratic equation $ax^2+bx+c=0$



2.6.2.3 Draw a flowchart to find the Fibonacci series till term≤1000.



2.7 ALGORITHM

Algorithm design is a specific method to create a mathematical process in solving problems. Applied algorithm design is algorithm engineering.

Algorithm design is identified and incorporated into many solution theories of operation research, such as dynamic programming and divide-and-conquer. Techniques for designing and implementing algorithm designs are algorithm design patterns, such as template method pattern and decorator pattern, and uses of data structures, and name and sort lists. Some current day uses of algorithm design can be found in internet retrieval processes of web crawling, packet routing and caching.

Mainframe programming languages such as ALGOL (for *Algo*rithmic *l*anguage), FORTRAN, COBOL, PL/I, SAIL, and SNOBOL are computing tools to implement an "algorithm design"... but, an "algorithm design" (a/d) is not a language. An algorithm design can be a hand written process, e.g. set of equations, a series of mechanical processes done by hand, an analog piece of equipment, or a digital process and/or processor.

One of the most important aspects of algorithm design is creating an algorithm that has an efficient runtime, also known as its Big O.

There are basically 5 fundamental techniques are used to design an algorithm efficiently:

Algorithm design techniques	**Examples**
Divide and Conquer	Binary search, Merge sort
Greedy Method	Knapsack problem, Minimum cost spanning tree problem (Kruskal's and Prim's Algorithm)

Dynamic Programming	All Pair Shortest Path Problem (Floyed Algorithm), Chain Matrix multiplication.
Backtracking	N-Queen's problem, Sum-of subset problem
Branch and Bound	Assignment problem, TSP (Travelling salesman problem)

2.7.1 Steps in development of Algorithms

1. Problem definition
2. Development of a model
3. Specification of Algorithm
4. Designing an Algorithm
5. Checking the correctness of Algorithm
6. Analysis of Algorithm
7. Implementation of Algorithm
8. Program testing
9. Documentation Preparation

2.7.2 Algorithms for Simple Problem

Example 2.7.2.1 : Develop an algorithm to find the average of three numbers

```
1. Start
2.  Read the numbers a, b, c
3. Compute the sum of a, b and c
4. Divide the sum by 3
5. Store the result in variable d
6. Print the value of d
7. Stop
```

Example 2.7.2.2 : Write an algorithm to find the largest of three numbers X, Y,Z.

Input: Three numbers a, b, and c
Output: x, the largest of a, b, and c

procedure max(a, b, c)

```
1. x := a
2. if b > x then
       x := b
3. if c>x then
       x := c
4. return(x)
5. end max
```

Example 2.7.2.3: Write an algorithm to check whether the given number is prime or not.

```
1. Start
2. Read number num
3. [ Initialize]
   i ← 2, flag ← 1
4. Repeat steps 4 through 6 until i < num or flag = 0
5. rem ← num mod i
```

```
6.  if rem = 0 then
          flag ← 0
    else
          i ← i + 1
7.  if flag = 0 then
          print “Prime”
    else
          print “not prime”
8.  Stop
```

2.8 PROGRAMMING LANGUAGES

Generally, we use languages like english, hindi, telugu etc., to make communication between two persons. That means, when we want to make communication between two persons we need a language through which persons can express their feelings. Similarly, when we want to make communication between user and computer or between two or more computers we need a language through which user can give information to computer and vice versa. When user wants to give any instruction to the computer the user needs a specific language and that language is known as computer language.

User interacts with the computer using programs and that programs are created using computer programming languages like C, C++, Java etc.,

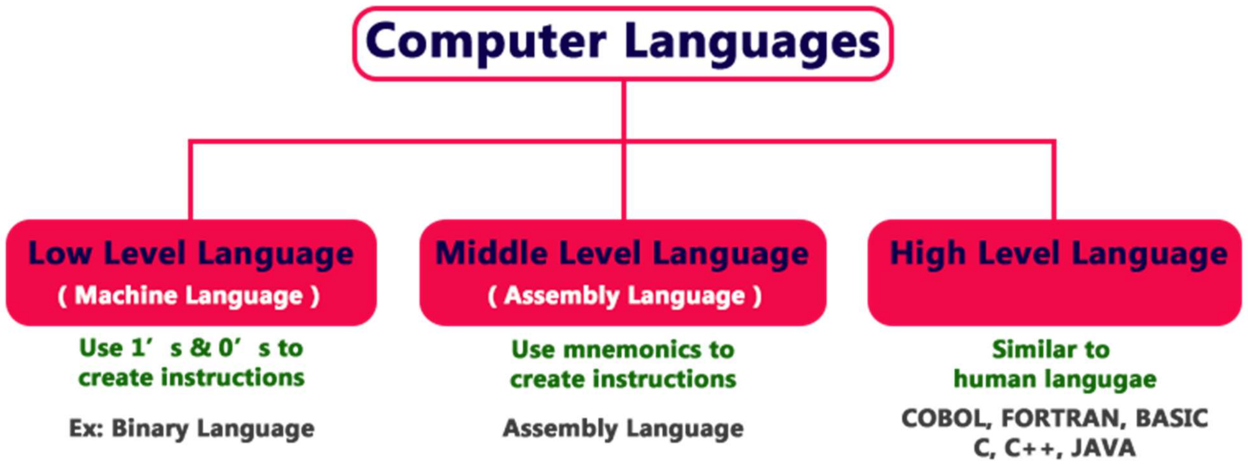
Computer languages are the languages through which user can communicate with the computer by writing program instructions.

Every computer programming language contains a set of predefined words and a set of rules (syntax) that are used to create instructions of a program.

The instructions in a programming language reflect the operations a computer can perform:
- A computer can transfer data from one place to another.
- A computer can get data from an input device (a keyboard or mouse, for example) and write data to an output device (a screen, for example).
- A computer can store data into and retrieve data from its memory and secondary storage (parts of a computer that we discuss in the next section).
- A computer can compare data values for equality or inequality and make decisions based on the result.
- A computer can perform arithmetic operations (addition and subtraction, for example) very quickly.
- A computer can branch to a different section of the instructions.

2.8.1 CLASSIFICATION OF PROGRAMMING LANGUAGE

Over the years, computer languages have been evolved from Low Level to High Level Languages. In the earliest days of computers, only Binary Language was used to write programs. The computer languages are classified as follows...



2.8.1.1 Low Level Language (Machine Language)

In a computer, data is represented electronically by pulses of electricity. Electric circuits, in their simplest form, are either on or off. Usually, a circuit that is on represents the number 1; a circuit that is off represents the number 0. Any kind of data can be represented by combinations of enough 1s and 0s. We simply have to choose which combination represents each piece of data we are using. For example, we could arbitrarily choose the pattern 1101000110 to represent the word *Basic*.
Data is represented by 1s and 0s in binary form. The binary (base–2) number system uses only 1s and 0s to represent numbers. (The decimal (base–10) number system uses the digits 0 through 9.) The word *bit* (short for binary digit) often is used to refer to a single 1 or 0. So the pattern 1101000110 has 10 bits. A binary number with 10 bits can represent 210 (1,024) different patterns. A byte is a group of eight bits; it can represent 28 (256) patterns. Inside the computer, each character (such as the letter A, the letter g, or a question mark) is usually represented by a byte.1 Groups of 16, 32, and 64 bits are generally referred to as words.

Low Level language is the only language which can be understood by the computer. Binary Language is an example of low level language. Low level language is also known as Machine Language. The binary language contains only two symbols 1 & 0. All the instructions of binary language are written in the form of binary numbers 1's & 0's. A computer can directly understand the binary language. Machine language is also known as Machine Code.

As the CPU directly understands the binary language instructions, it does not requires any translater. CPU directly starts executing the binary language instructions, and takes very less time to execute the instructions as it does not requires any translation. Low level language is considered as the First Generation Language (1GL).

Advantages

- A computer can easily understand the low level language.
- Low level language instructions are executed directly without any translation.
- Low level language instructions require very less time for thier execution.

Disadvantages

- Low level language instructions are very difficult to use and understand.
- Low level language instructions are machine dependent, that means a program written for a particular machine does not executes on other machine.
- In low level language, there is more chance for errors and it is very difficult to find errors, debug and modify.

2.8.1.2 Middle Level Language (Assembly Language)

Middle level language is a computer language in which the instructions are created using symbols such as letters, digits and special characters. Assembly language is an example of middle level language. In assembly language, we use predefined words called mnemonics. Binary code instructions in low level language are replaced with mnemonics and operands in middle level language. But computer cannot understand mnemonics, so we use a translator called Assembler to translate mnemonics into binary language. Assembler is a translator which takes assembly code as input and produces machine code as output. That means, computer cannot understand middle level language, so it needs to be translated into low level language to make it understandable by the computer. Assembler is used to translate middle level language to low level language.

Advantages

- Writing instructions in middle level language is easier than writing instructions in low level language.
- Middle level language is more readable compared to low level language.
- Easy to understand, find errors and modify.

Disadvantages

- Middle level language is specific to a particular machine architecture, which means it is machine dependent.
- Middle level language needs to be translated into low level language.
- Middle level language executes slower compared to low level language.

2.8.1.3 High Level Language

High level language is a computer language which can be understood by the users. High level language is very similar to the human languages and have a set of grammar rules that are used to make instructions more easily.

Every high level language have a set of predefined words known as Keywords and a set of rules known as Syntax to create instructions. High level language is easier to understand for the users but the computer cannot understand it. High level language needs to be converted into low level language to make it understandable by the computer. We use Compiler or interpreter to convert high level language to low level language.

Languages like COBOL, FORTRAN, BASIC, C, C++, JAVA etc., are the examples of high level languages. All these programming languages use human understandable language like english to write program instructions. These instructions are converted to low level language by the compiler so that it can be understood by the computer.

Advantages

- Writing instructions in high level language is easier.
- High level language is more readable and understandable.
- The programs created using high level language runs on different machines with little change or no change.
- Easy to understand, create programs, find errors and modify.

Disadvantages

- High level language needs to be translated to low level language.
- High level language executes slower compared to middle and low level languages.

2.9 Program Development Environments

The environment under which a program is designed, coded, tested & debugged is called Host Environment. The external environment which supports the execution of a program is termed as Operating or Target Environment. Host and Target environment may be different for a program or application.

2.9.1 Programming Environments (Host Environment)

It is the environment in which programs are created and tested. It tends to have less influence on language design than the operating environment in which programs are expected to be executed. The production of programs that operate reliably and efficiently is made much simpler by a good programming environment and by a language that allows the use of good programming tools and practices.

2.9.2 Target Environments

Target environments can be classified into 3 categories – Batch Processing Environment, Interactive Environment, and Embedded System Environment. Each poses different requirement on languages adapted for those environments.

2.9.3 Batch-Processing Environments

In batch-processing environments, the input data are collected in ‘batches’ on files and are processed in batches by the program. For example, the backup process on an organization. The transaction details of all the departments are collected for backup at one place and the backup is done at a time at the end of the day.

2.9.4 Interactive Environments

In interactive environment, a program interacts directly with a user at a display console, by alternately sending output to the display & receiving input from the keyboard or mouse. Examples include database management systems, word processing systems etc.

2.9.5 Embedded System Environments

An embedded computer system is used to control part of a larger system such as an industrial plant (computerized machineries) or an aircraft. The computer system will be an integral part of the larger system, failure of which would imply failure of the larger system as well.

2.10 TRANSLATOR

A program written in high-level language is called as source code. To convert the source code into machine code, translators are needed.
A translator takes a program written in source language as input and converts it into a program in target language as output.
It also detects and reports the error during translation.

Roles of translator are:

• Translating the high-level language program input into an equivalent machine language program.
• Providing diagnostic messages wherever the programmer violates specification of the high-level language program.

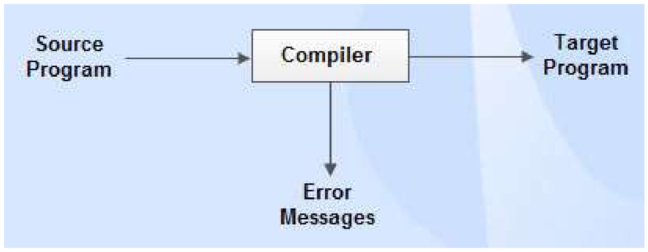
Different type of translators

The different types of translator are as follows:

2.10.1 Compiler

Compiler is a translator which is used to convert programs in high-level language to low-level language. It translates the entire program and also reports the errors in source program encountered during the translation.

It is a program which translates a high level language program into a machine language program. A compiler is more intelligent than an assembler. It checks all kinds of limits, ranges, errors etc. But its program run time is more and occupies a larger part of the memory. It has slow speed. Because a compiler goes through the entire program and then translates the entire program into machine codes. If a compiler runs on a computer and produces the machine codes for the same computer then it is known as a self compiler or resident compiler. On the other hand, if a compiler runs on a computer and produces the machine codes for other computer then it is known as a cross compiler.



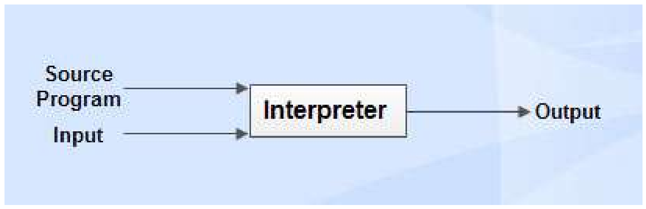
2.10.2 Interpreter

Interpreter is a translator which is used to convert programs in high-level language to low-level language. Interpreter translates line by line and reports the error once it encountered during the translation process.

An interpreter is a program which translates statements of a program into machine code. It translates only one statement of the program at a time. It reads only one statement of program, translates it and executes it. Then it reads the next statement of the program again translates it and executes it. In this way it proceeds further till all the statements are translated and executed. On the other hand, a compiler goes through the entire program and then translates the entire program into machine codes. A compiler is 5 to 25 times faster than an interpreter.

By the compiler, the machine codes are saved permanently for future reference. On the other hand, the machine codes produced by interpreter are not saved. An interpreter is a small program as compared to compiler. It occupies less memory space, so it can be used in a smaller system which has limited memory space.

It directly executes the operations specified in the source program when the input is given by the user. It gives better error diagnostics than a compiler.



2.10.3 Differences between compiler and interpreter

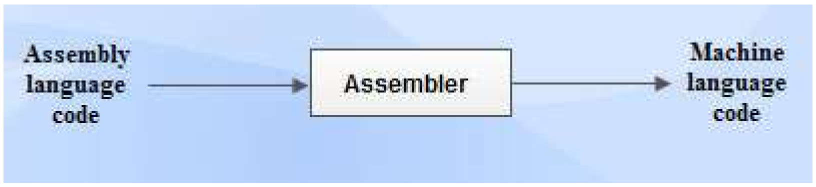
SI. No	**Compiler**	**Interpreter**
1	Performs the translation of a program as a whole.	Performs statement by statement translation.
2	Execution is faster.	Execution is slower.
3	Requires more memory as linking is needed for the generated intermediate object code.	Memory usage is efficient as no intermediate object code is generated.
4	Debugging is hard as the error messages are generated after scanning the entire program only.	It stops translation when the first error is met. Hence, debugging is easy.
5	Programming languages like C, C++ uses compilers.	Programming languages like Python, BASIC, and Ruby uses interpreters.

2.10.4 Assembler

Assembler is a translator which is used to translate the assembly language code into machine language code.

A computer will not understand any program written in a language, other than its machine language. The programs written in other languages must be translated into the machine language. Such translation is performed with the help of software. A program which translates an assembly language program into a machine language program is called an assembler. If an assembler which runs on a computer and produces the machine codes for the same computer then it is called self assembler or resident assembler. If an assembler that runs on a computer and produces the machine codes for other computer then it is called Cross Assembler.

Assemblers are further divided into two types: One Pass Assembler and Two Pass Assembler. One pass assembler is the assembler which assigns the memory addresses to the variables and translates the source code into machine code in the first pass simultaneously. A Two Pass Assembler is the assembler which reads the source code twice. In the first pass, it reads all the variables and assigns them memory addresses. In the second pass, it reads the source code and translates the code into object code.



2.11 Linker

In high level languages, some built in header files or libraries are stored. These libraries are predefined and these contain basic functions which are essential for executing the program. These functions are linked to the libraries by a program called Linker. If linker does not find a library of a function then it informs to compiler and then compiler generates an error. The compiler automatically invokes the linker as the last step in compiling a program.

Not built in libraries, it also links the user defined functions to the user defined libraries. Usually a longer program is divided into smaller subprograms called modules. And these modules must be combined to execute the program. The process of combining the modules is done by the linker.

2.12 Loader

Loader is a program that loads machine codes of a program into the system memory. In Computing, a **loader** is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory. Once loading is complete, the operating system starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders. In many operating systems the loader is permanently resident in memory.

2.13 STRUCTURED PROGRAMMING CONCEPT

Structured programming is a technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure. Three types of control flow are used: sequential, test, and iteration.

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures, for and while loops—in contrast to using simple tests and jumps such as the *go to* statement which could lead to "spaghetti code" causing difficulty to both follow and maintaing.

Programming languages require that we use certain *control structures* to express algorithms as source code. There are four basic ways of structuring statements (instructions) in most programming languages: by sequence, selection, loop, and with subprograms.

A *sequence* is a series of statements that are executed one after another.
Selection, the conditional control structure, executes different statements depending on certain conditions.
The repetitive control structure, the *loop*, repeats statements while certain conditions are met.
The *subprogram* allows us to structure our code by breaking it into smaller units.
Each of these ways of structuring statements controls the order in which the computer executes the statements, which is why they are called control structures.

Assume you're driving a car. Going down a straight stretch of road is like following *sequence* of instructions. When you come to a fork in the road, you must decide which way to go and then take one or the other branch of the fork. This is what the computer does when it encounters a *selection control structure* (sometimes called a *branch* or *decision*) in a program. Sometimes you have to go around the block several times to find a place to park. The computer does the same sort of thing when it encounters a *loop* in a program.
A *subprogram* is a named sequence of instructions written separately from the main program. When the program executes an instruction that refers to the name of the subprogram, the code for the subprogram is executed. When the subprogram has finished executing, execution of the program resumes at the next instruction. Suppose, for example, that every day you go to work at an office. The directions for getting from home to work form a procedure called "Go to the office." It makes sense, then, for someone to give you directions to a meeting by saying "Go to the office, then go four blocks west," without listing all the steps you have to take to get to the office. Subprograms allow us to write parts of our programs separately and then assemble them into final form. They can greatly simplify the task of writing large programs.

2.14 Summary

- Computer programming is a process that leads from an original formulation of a computing problem to executable computer programs.
- Programming involves activities such as analysis, developing understanding, generating algorithms, verification of requirements of algorithms including their correctness and resources consumption, and implementation of algorithms in a target programming language.
- Programs consist of one or more statements. Statements consist of one or more expressions. Expressions consist of one or more operands in conjunction with one or more operators.
- A good computer program should have following characteristics – Portability, Readability, Efficiency, Structural, Flexibility, Generality and Documentation.
- A Program Development Life Cycle contains 6 phases:- Problem Definition, Problem Analysis, Algorithm Development. Coding & Documentation, Testing & Debugging and Maintenance.

- Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm which omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines.
- A flowchart is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.
- Algorithm design is a specific method to create a mathematical process in solving problems. There are basically 5 fundamental techniques are used to design an algorithm efficiently: Divide and Conquer, Greedy Method, Dynamic Programming, Backtracking, Branch and Bound.
- User interacts with the computer using programs and that programs are created using computer programming languages like C, C++, Java etc.
- Low Level language is the only language which can be understood by the computer. Binary Language is an example of low level language. Low level language is also known as Machine Language.
- Middle level language is a computer language in which the instructions are created using symbols such as letters, digits and special characters. Assembly language is an example of middle level language.
- High level language is a computer language which can be understood by the users. High level language is very similar to the human languages and have a set of grammar rules that are used to make instructions more easily.
- The environment under which a program is designed, coded, tested & debugged is called Host Environment. The external environment which supports the execution of a program is termed as Operating or Target Environment.
- A program written in high-level language is called as source code. To convert the source code into machine code, translators are needed.
- Compiler is a translator which is used to convert programs in high-level language to low-level language. It translates the entire program and also reports the errors in source program encountered during the translation.
- Interpreter is a translator which is used to convert programs in high-level language to low-level language. Interpreter translates line by line and reports the error once it encountered during the translation process.
- Assembler is a translator which is used to translate the assembly language code into machine language code.
- Loader is a program that loads machine codes of a program into the system memory while linker is used to link the library functions.
- Structured programming is a technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point.

2.15 Questions for Exercises

1 What do you mean by Computer Programming? What are the characteristics of a good computer program?
2 Explain various phases of Programming Life Cycle.
3 What do you mean by pseudocode? What are the rules for writing a pseudocode?
4 Write a pseudocode to display a series of first 100 natural numbers.
5 Define a flow chart with all the symbols used in designing it.
6 Draw a flow chart to print first 10 prime numbers.
7 Explain various Algorithm design techniques.
8 Write down the classification of Computer Languages.
9 What are Translators? Explain in details.
10 What do you mean by Linkers and Loaders?

2.16 Suggested Readings

- ***Algorithms: Design and Analysis****, Harsh Bhasin, 2015*
- ***The Design and Analysis of Computer Algorithms, 6th Edition,*** *Alfred V Aho, John E Hopcroft, 2014*
- **How to Solve it by Computer,** *6th Edition***,** *R. G, Dromey, 2001*
- ***Programming Languages*** **(*Second Edition*) ─ *Concepts and Constructs,*** *Ravi Sethi: (Pearson Education, Asia, 1996).*