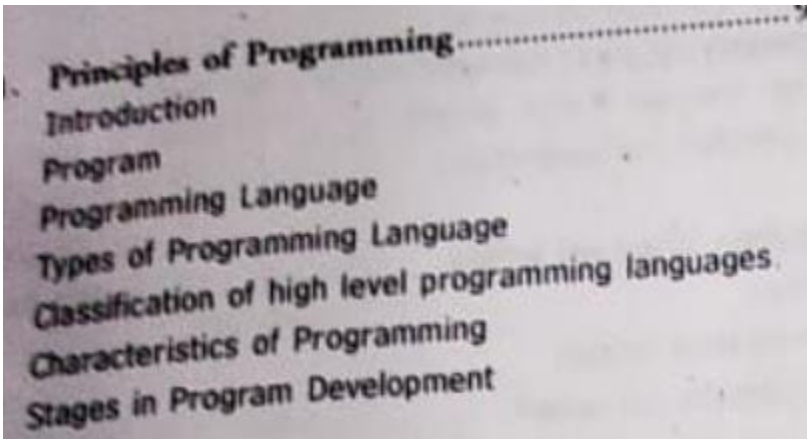
BSC FY NOTES

UNIT – 1



What is a program?

A program is a set of logically related instructions that is arranged in a sequence that directs the computer in solving a problem.

Programming - Programming is the implementation of logic to facilitate specified computing operations and functionality. It occurs in one or more languages, which differ by application, domain and programming model

A GOOD PROGRAM?

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:

1. Portability: Portability refers to the ability of an application to run on different platforms (operating systems + computer hardware) 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.

2. 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.
3. 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.
4. Structural Modularity: 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.
5. Flexibility: A program should be flexible enough to handle most of the changes without having to rewrite the entire program or parts of it. 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.
6. 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.
7. 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.

What is a programming language?

A programming language defines a set of instructions that are compiled together to perform a specific task by the CPU (Central Processing Unit). The programming language mainly refers to high-level languages such as C, C++, Pascal, Ada, COBOL, etc.

A GOOD Programming Language?

A good programming language must be simple and easy to learn and use.It should provide a programmer with a clear, simple and unified set of concepts that can be grasped easily. The overall simplicity of a this strongly affects the readability of the programs written in that language and programs that are easier to read and understand are easier to maintain. It is also easy to develop and implement a compiler or an interpreter for a simple language. However,the power needed for the language should not be sacrificed for simplicity. For Example, BASIC is liked by many programmers because of its simplicity.

1-Naturalness:

A good language should be natural for the application area for which it is designed. That is, it should provide appropriate operators, data structures, control structures and a natural syntax to facilitate programmers to code their problems easily and efficiently. FORTRAN and COBOL are good examples of languages possessing high degree of naturalness in scientific and business application areas, respectively.

2-Abstraction:

Abstraction means ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored. The degree of

abstraction allowed by a language directly affects its ease of programming. For Example, object-oriented languages support high degree of abstraction. Hence, writing programs in object-oriented languages is much easier. Object-oriented also support re usability of program segments due to this feature.

3-Efficiency:
Programs written in a good language are translated into machine code efficiently, are executed and require relatively less space in memory. That is, a good programming language is supported with a good language translator (a compiler or an interpreter) that gives due consideration to space and time efficiency.

4-Structured Programming Support:

A good language should have necessary features to allow programmers to write their programs based on the concepts of structured programming. This property greatly affects the ease with which a program may be written., tested and maintained. Moreover, it forces a programmer to look at a problem in a logical way so that fewer errors are created while writing a program for the problem.

5-Compactness:

In a good language, programmers should be able to express the intended operations concisely without losing readability. Programmers generally do not like a verbose language because they need to write too much. Many programmers dislike COBOL, because it is verbose in nature (Lacks Compactness)

6-Locality:

A good language should be such that while writing a program, a programmer need not jump around the visually as the text of a program is prepared. This allows the programmer to concentrate almost solely on the part of the program around the statement currently being worked with. COBOL and to some extent C and Pascal lack locality because data definitions are separated from processing statements, perhaps by many pages of code, or have to appear before any processing statement in the function/procedure.

7-Extensibility:

A good language should also allow extensions through a simply, natural and elegant mechanism. Almost all languages provide subprogram definition mechanisms for the purpose, but some languages are weak in this aspect.

8-Suitability to its Environment:

Depending upon the type of application for which a programming language has been designed, the language must also be made suitable to its environment. For Example, a language designed for a real-time application must be interactive in nature. On the other hand, languages used for data-processing jobs like payroll, stores accounting etc. may be designed to operative in batch mode.

Classification of Programming Languages

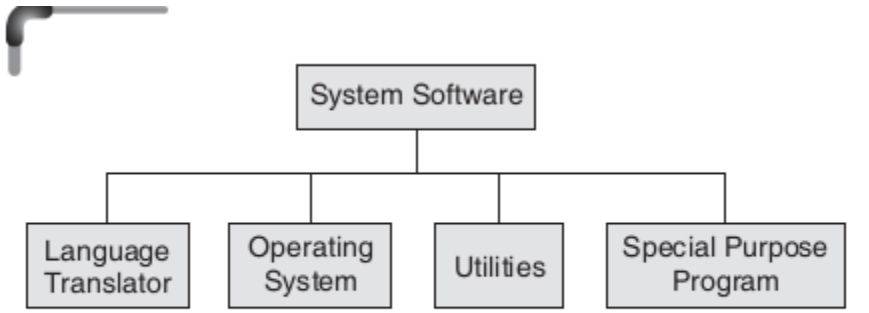


Figure 1.2 Categories of system software

Broad Classification -

System Programming Languages

System programs or software are designed to make the computer easier to use. An example of system software is an operating system consisting of many other programs that control input/output devices, memory, processor, schedule the execution of multiple tasks, etc. To write an operating system program, the programmer needs instructions to control the computer's circuitry as well as manage the resources of the computer. For example, instructions that move data from one location of storage to a register of the processor are required. Assembly language, which has a one-to-one correspondence with machine code, was the

normal choice for writing system programs like operating systems. But today C is widely used to develop system software.

Application Programming Languages

There are two main categories of application programs: business programs and scientific application programs. Application programs are designed for specific computer applications, such as payroll processing and inventory control. To write programs for payroll processing or other such applications, the programmer does not need to control the basic circuitry of a computer. Instead, the programmer needs instructions that make it easy to input data, produce output, perform calculations, and store and retrieve data. Programming languages suitable for such application programs have the appropriate instructions. Most programming languages are designed to be good for one category of applications but not necessarily for the other, although there are some general-purpose languages that support both types. Business applications are characterized by processing of large inputs and high-volume data storage and retrieval but call for simple calculations. Languages which are suitable for business program development must support high-volume input, output, and storage but do not need to support complex calculations. On the other hand, programming languages designed for writing scientific programs contain very powerful instructions for calculations but have poor instructions for input, output, etc. Among the traditionally used programming languages, COBOL is more suitable for business applications whereas FORTRAN is more suitable for scientific applications.

1.2.3 Low-level Languages

A low-level computer programming language is one that is closer to the native language of the computer, which is 1's and 0's.

Machine language

This is a sequence of instructions written in the form of binary numbers consisting of 1's and 0's to which the computer responds directly. The machine language is also referred to as the machine code, although the term is used more broadly to refer to any program text.

A machine language instruction generally has three parts as shown in Fig. 1.5. The first part is the command or operation code that conveys to the computer what function

has to be performed by the instruction. All computers have operation codes for functions such as adding, subtracting and moving. The second part of the instruction either specifies that the operand contains data on which the operation has to be performed or it specifies that the operand contains a location, the contents of which have to be subjected to the operation.

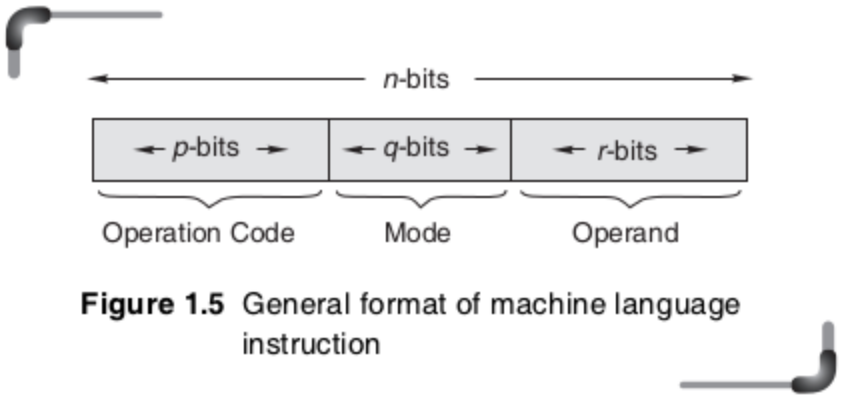


Figure 1.5 General format of machine language instruction

Just as hardware is classified into generations based on technology, computer languages also have a generation classification based on the level of interaction with the machine. Machine language is considered to be the *first generation language* (1GL).

Advantage of machine language The CPU directly understands machine instructions, and hence no translation is required. Therefore, the computer directly starts executing the machine language instructions, and it takes less execution time.

Disadvantages of machine language

- ***Difficult to use*** It is difficult to understand and develop a program using machine language. For anybody checking such a program, it would be difficult to forecast the output when it is executed. Nevertheless, computer hardware recognizes only this type of instruction code.

- ***Machine dependent*** The programmer has to remember machine characteristics while preparing a program. As the internal design of the computer is different across types, which in turn is determined by the actual design or construction of the ALU, CU, and size of the word length of the memory unit, the machine language also varies from one type of computer to another. Hence, it is important to note that after becoming proficient in the machine code of a particular

computer, the programmer may be required to learn a new machine code and would have to write all the existing programs again in case the computer system is changed.

- ***Error prone*** It is hard to understand and remember the various combinations of 1's and 0's representing data and instructions. This makes it difficult for a programmer to concentrate fully on the logic of the problem, thus frequently causing errors.
- ***Difficult to debug and modify*** Checking machine instructions to locate errors are about as tedious as writing the instructions. Further, modifying such a program is highly problematic.

Following is an example of a machine language program for adding two numbers.

Assembly language

When symbols such as letters, digits, or special characters are employed for the operation, operand, and other parts of the instruction code, the representation is called an assembly language instruction. Such representations are known as mnemonic codes; they are used instead of binary codes. A program written with mnemonic codes forms an assembly language program. This is considered to be a *second generation language* (2GL).

Machine and assembly languages are referred to as low-level languages since the coding for a problem is at the individual instruction level. Each computer has its own assembly language that is dependent upon the internal architecture of the processor.

An *assembler* is a translator that takes input in the form of the assembly language program and produces machine language code as its output. An instruction word consists of parts shown in Fig. 1.5 where,

- the Opcode (Operation Code) part indicates the operation to be performed by the instruction and
- the mode and operand parts convey the address of the data to be found or stored.

The following is an example of an assembly language program for adding two numbers *X* and *Y* and storing the result in some memory location.

From this example program, it is clear that using mnemonics such as LD, ADD, and HALT, the readability of the program has improved significantly.

An assembly language program cannot be executed by a machine directly as it is not in a binary machine language form. An assembler is needed to translate an assembly language program into the object code, which can then be executed by the machine. The object code is the machine language code. This is illustrated in Fig. 1.6.

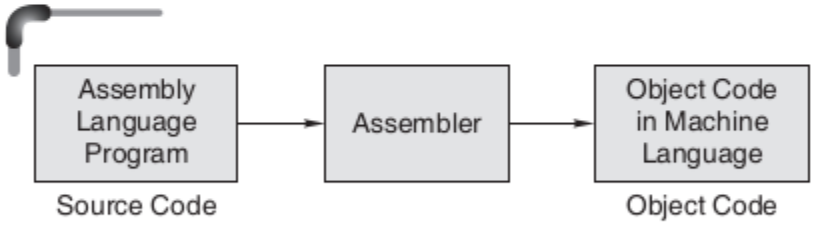


Figure 1.6 Assembler

Advantage of assembly language Writing a program in assembly language is more convenient than writing one

in machine language. Instead of binary sequence, as in machine language, a program in assembly language is written in the form of symbolic instructions. This gives the assembly language program improved readability.

Disadvantages of assembly language

- Assembly language is specific to a particular machine architecture, i.e., machine dependent. Assembly languages are designed for a specific make and model of a microprocessor. This means that assembly language programs written for one processor will not work on a different processor if it is architecturally different. That is why an assembly language program is not portable.
- Programming is difficult and time consuming.
- The programmer should know all about the logical structure of the computer.

1.2.4 High-level Languages

High-level programming languages such as COBOL, FORTRAN, and BASIC were mentioned earlier in the chapter. Such languages have instructions that are similar to human languages and have a set grammar that makes it easy for a programmer to write programs and identify and correct errors in them. To illustrate this point, a program written in BASIC, to obtain the sum of two numbers, is shown below.

The time and cost of creating machine and assembly language programs were quite high. This motivated the development of high-level languages.

Advantages of high-level programming languages

Readability Programs written in these languages are more readable than those written in assembly and machine languages.

Advantages of high-level programming languages

Readability Programs written in these languages are more readable than those written in assembly and machine languages.

Portability High-level programming languages can be run on different machines with little or no change. It is, therefore, possible to exchange software, leading to creation of program libraries.

Easy debugging Errors can be easily detected and removed.

Ease in the development of software Since the commands of these programming languages are closer to the English language, software can be developed with ease.

High-level languages are also called *third generation languages* (3GLs).

Low-level language	High-level language
It is a machine-friendly language, i.e., the computer understands the machine language, which is represented in 0 or 1.	It is a user-friendly language as this language is written in simple English words, which can be easily understood by humans.
The low-level language takes more time to execute.	It executes at a faster pace.
It requires the assembler to convert the assembly code into machine code.	It requires the compiler to convert the high-level language instructions into machine code.
The machine code cannot run on all machines, so it is not a portable language.	The high-level code can run all the platforms, so it is a portable language.
It is memory efficient.	It is less memory efficient.
Debugging and maintenance are not easier in a low-level language.	Debugging and maintenance are easier in a high-level language.

High-Level Language

The high-level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer. The high-level languages are considered as high-level because they are closer to human languages than machine-level languages.

When writing a program in a high-level language, then the whole attention needs to be paid to the logic of the problem.

A compiler is required to translate a high-level language into a low-level language.

Advantages of a high-level language

The high-level language is easy to read, write, and maintain as it is written in English like words.

The high-level languages are designed to overcome the limitation of low-level language, i.e., portability. The high-level language is portable; i.e., these languages are machine-independent.

Classification of High-Level Programming Languages

The High-Level Language is basically classified into 3 types namely:

1. ***Procedural Language***
2. ***Functional Language***
3. ***Object Oriented Language***

Procedural Language:

The most common high-level languages today are procedure-oriented languages. In these languages, one or more related blocks of statements that perform some complete functions are grouped together into a program module, or procedure, & given a name such as "Procedure A." If the same sequence of operations is needed elsewhere in the program, a simple statement can be used to refer back to the procedure. In essence, a procedure is just a mini-program. A large program can be constructed by grouping together procedures that perform different tasks.

Procedural languages allow programs to be shorter and easier for the computer to read, but they require the programmer to design each procedure to be general enough to be used in different situations.

Functional Language:

Functional languages treat procedures like mathematical functions and allow them to be processed like any other data in a program. This allows a much higher and more rigorous level of program construction. Functional languages also allow variables—symbols for data that can be specified and changed by the user as the program is running—to be given values only once. This simplifies programming by reducing the need to be concerned with the exact order of statement execution, since a variable does not have to be redeclared, or restated, each time it is used in a program statement. Many of the ideas from functional languages have become key parts of many modern procedural languages.

Object Oriented Language:

Object-oriented languages are outgrowths of functional languages. In object-oriented languages, the code used to write the program and the data processed by the program are grouped together into units called objects. Objects are further grouped into classes, which define the attributes objects must have.

A simple example of a class is the class Book.

Objects within this class might be Novel and Short Story. Objects also have certain functions associated with them, called methods. The computer accesses an object through the use of one of the object's methods. The method performs some action to the data in the object and returns this value to the computer. Classes of objects can also be further grouped into hierarchies, in which objects of one class can inherit methods from another class. The structure provided in object-oriented languages makes them very useful for complicated programming tasks

Stages of program development process

The various stages in the development of a computer program are:

1. Problem Definition
2. Program Design
3. Coding
4. Debugging
5. Testing
6. Documentation
7. Maintenance

Problem Definition:

- The first step in the process of program development is the thorough understanding and identification of the problem for which is the program or software is to be developed.
- In this step the problem has to be defined formally.
- All the factors like Input/output, processing requirement, memory requirements, error handling, interfacing with other programs have to be taken into consideration in this stage.

Program Design:

- The next stage is the program design. The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.
    - Algorithm
    - Flowchart

Coding:

- Once the design process is complete, the actual computer program is written, i.e., the instructions are written in a computer language.
- Coding is generally a very small part of the entire program development process and also a less time-consuming activity in reality.
- In this process all the syntax errors i.e., errors related to spelling, missing commas, undefined labels etc. are eliminated.
- For effective coding some of the guide lines which are applied are:
    - Use of meaningful names and labels of variables,
    - Simple and clear expressions,
    - Modularity with emphasis on making modules generalized,
    - Making use of comments and indenting the code properly,
    - Avoiding jumps in the program to transfer control.

Debugging:

- At this stage the errors in the programs are detected and corrected.
- This stage of program development is an important process. Debugging is also known as program validation.
- Some common errors which might occur in the programs include:
    - Un initialization of variables.
    - Reversing of order of operands.
    - Confusion of numbers and characters.
    - Inverting of conditions eg jumping on zero instead of on not zero.

Testing:

- The program is tested on a number of suitable test cases.
- A test plan of the program has to be done at the stage of the program design itself.
- This ensures a thorough understanding of the specifications.
- The most trivial and the most special cases should be identified and tested.
- It is always useful to include the maximum and minimum values of all variables as test data.

Documentation:

- Documentation is a very essential step in the program development.
- Documentation helps the users and the people who maintain the software.
- This ensures that future modification if required can be done easily. Also, it is required during redesigning and maintenance.

Maintenance:

- Updating and correction of the program for changed conditions and field experience is accounted for in maintenance.
- Maintenance becomes essential in following situations:
    - Change in specification,
    - Change in equipment,
    - Errors which are found during the actual execution of the program.