## Week 01

### Computer Programming:

A computer program is a collection of instructions that performs a specific task when executed by a computer.

A computer program is usually written by a computer programmer in a programming language.

Programming can be classified into two types: *systems programming* and *(application) programming*.

*System programming* is handled by computer engineers or system programmers to develop system programs such as operating systems, language translators, etc.

*(Application) programming* is handled by ordinary computer programmers to develop application software such as payroll system, stock control system, etc.

The possibility of programming a task by a machine is called *computability*.

A program should produce the required result for all possible inputs; and it is called *correctness*.

Programming that produces programs with clean flow, clear design, and a degree of modularity or hierarchical structure is called *structured programming*.

The three basic constructs in structured programming are *sequence* (ordered set of statements), *selection* (conditional branch), and *iteration* (repletion). There is no GOTO statement to jump to any place in a structured program.

An approach to programming in which the program is broken into several independently compiled modules is called *modular programming*. It facilitates group programming efforts.

An approach to programming that implements a program in top-down fashion is referred to as *top-down programming*. Typically this is done by writing a main body which calls to several major routines (implemented as *stubs*). Each routine is then coded, calling other, lower-level routines (also done initially as stubs). Here, *stub* is a placeholder for a routine to be written later.

*Bottom-up programming* is a programming technique in which lower-level functions are developed and tested first; higher-level functions are then built using the lower-level functions.

### Programming Language:

*Programming Language* is any artificial language that can be used to define a sequence of instructions that can ultimately be processed and executed by the computer.

A program called, *language translator*, translates statements written in one programming language into their machine language.

**Levels/Generations of Languages:**

Languages are generally divided into five levels/generations:

- *Machine language*
- *Assembly language*
- *High-level languages*
- *Very high-level languages*
- *Natural languages*.

Programs that are written in 0s and 1s are in the actual *machine language*. Machine code consists of binary coded instructions and data intermixed. All other languages must be translated into machine language before execution. Each type of computer has its own unique machine langue. Machine language is very difficult to write programs and to detect and correct program errors. A thorough knowledge of computer hardware is required for machine language programming. Here, programmers cannot concentrate on data processing problems, because more attention should be paid for the architecture of the computer system. Machine language programs are not portable among different types of computers. However, machine language programs can be executed much faster and maximum usage of resources (e.g. memory) can be obtained.

*Assembly language* is the direct symbolic representation of a machine language. Assembly languages include *IBM, BAL* and *VAX Macro*. Assembly languages use English-like mnemonic codes to represent the operations: *A* for add, *C* for compare, *MP* for multiply, *STO* for storing information in memory, and so on. Assembly language specifies registers with meaningful codes, such as AX, BX, etc., instead of binary numbers. Furthermore, it permits the use of names – perhaps RATE or TOTAL – for memory locations, instead of the actual binary memory address. As with machine language, each type of computer has its own unique assembly language. Therefore, both machine and assembly languages are considered as *machine oriented* or *low-level computer languages*. Assembly language is easy to write programs and to detect and correct program errors compared to machine language programs. As source program should be translated into machine language, more time is required for execution.

To overcome problems of low-level languages, *high-level languages* were developed. As standard words of those languages are closer to human language (English) it is much easier to write programs and to detect and correct errors. As high-level languages are machine independent, programs are portable, less hardware knowledge is required, and more attention can be paid for data processing requirements. Here, a lesser number of instructions are required. Due to the compilation process more time is needed to execute the program. Here,

computer resources cannot be used fully. *FORTRAN (Formula translator), ALGOL (Algorithmic Language), COBOL (Common business oriented language), RPG (Report program generator), BASIC (Beginners all purpose symbolic instruction code), Pascal, C, C++, Java,* and *Visual Basic,* are some of the examples for high-level language. For any given high-level language, there are usually a number of translators available, each of which translates a program in that language to the machine language for a specific type of computer. *Very high-level languages* are often known as *4GLs (fourth-generation languages)*. Languages belonging to the first three generations are *procedural languages*, consisting of instructions that describe the step-by-step procedure to solve the problem. 4GLs are *nonprocedural languages*, in which the programmer specifies the desired results, and the language develops the solution. *Query languages*, such as *SQL (Structured query language)*, are variations on 4GLs and are used to retrieve information from databases.

The *natural language* (sometimes considered to be *5GLs*) translates human instructions into code the computer can execute. If it does not understand the user's request, it politely asks for further explanation. For example, *INTELLECT*, a natural language, would use a statement like, "What are the average exam scores in MIS?".

E.g.: Comparison among different levels of programming languages.

| *Machine Language* | *Assembly Language* | *High-Level language* | *4GL / Natural Language* |
|---|---|---|---|
| 1010 11001<br>1011 11010<br>1100 11011 | LOD Y<br>ADD Z<br>STR X | X = Y + Z | SUM THE FOLLOWING NUMBERS |

**Language Translators:**

All the source programs should be converted into their machine language before the execution process. *Language translators* are used for this translation.

There are three types of language translators: *Assemblers, Compilers,* and *Interpreters*.

*Assembler* is a language translator which translates source programs written in assembly language into their object programs.

*Compiler* translates/compiles the entire program written in a high-level language into its object program considering the program as a single unit. After the compilation process the source program is no more required as the object program is available. A compiler is able to look at the whole of a program and to work out the best way of translating. This is sometimes called *optimization* and is one of the reasons why a compiler can produce efficient code. Once all errors have been corrected, the program will run faster. However errors are more difficult to find during compilation because they are not reported until the end of the process and a single error can generate additional error messages making it harder to locate the error. When the

program is in testing mode, there is no benefit to having a stored executable program since this will have to be rebuilt each time.

*Interpreter* is a language translator which converts high-level language statements into equivalent machine language statements one at a time as the program is executed. It does not generate an object program as a separate unit. Here, each statement is checked for syntax, then converted into machine code, and then executed. The presence of the source program is required for the execution of the program and it should be interpreted each time. Here, in the case of repetition, some program statements may be translated many times. Executing an interpreted program is slower than executing a compiled program since each statement has to be converted to binary each time even if it has been converted previously. With an interpreted language there is always the danger that a syntax error may exist in a section of code that has not been tested. However, there is no lengthy compile and link cycle. If the program encounters an error, the interpretations stops and an error message is displayed so the user can correct it. Therefore, interpreters are comfortable for beginners because small programs can be written and tested very quickly.

Most current languages come in a comprehensive package called an *integrated development environment (IDE)* that includes language aware editor, project build capability (compile and link), debugger, and other programming tools. Other programming tools include diagramming packages, code generators, libraries of reusable objects and program code, and prototyping tools.

**Some Commonly Used High-Level Languages:**

*FORTRAN (Formula translator)* was developed by IBM and introduced in 1954. It was the first high-level language. It is very good at representing complex mathematical formulas.

*ALGOL (Algorithmic Language)* was the first structured procedural programming language developed in late 1950s.

*COBOL (Common business oriented language)* is a verbose (wordy), English-like compiled programming language developed between 1959 and 1961 and still in widespread use, especially in business applications typically run on mainframes. It is very good for processing large, complex data files and for producing well-formatted business reports.

*RPG (Report program generator)* was developed by IBM in 1965 to allow rapid creation of reports from data stored in the computer files.

*BASIC (Beginners All Purpose Symbolic Instruction Code)* was originally developed by Dartmouth College professors John Kemeny and Thomas Kurtz in 1965 to teach programming to their students.

*Pascal* was designed between 1967 and 1971 by Niklaus Wirth. It is a compiled, structured language built upon ALGOL, simplifies syntax while adding data types and structures such as sub ranges, enumerated data types, files, records, and sets.

*C* was created in 1972. One of the C's primary advantages is its portability: There are C compilers for almost every combination of computer and operating system available today.

*C++* is an *object-oriented* version of C, developed in the early 1980s.

*Java* is a network-friendly object-oriented programming language derived from C++ that permits a piece of software to run directly on many different platforms. Java programs are not compiled directly into machine language, but into an intermediate language called *bytecode*. This bytecode is then executed by a universal platform, called the *JVM (Java Virtual Machine)*, which sits atop a computers regular platform. The JVM interprets (translates) compiled Java code into instructions that the platform underneath can understand. Java provides high level of security to the user and the network. Java uses Unicode coding system, which allows displaying all character sets in a uniform manner. Web pages can include Java mini programs, called *applets*, which run on a Java platform included in the user's *Web browser*.

*Visual Basic (VB)* was introduced by Microsoft in 1987 as its first visual development tool. It allows the programmer to easily create complex user interfaces containing standard Windows features, such as buttons, dialog boxes, scrollbars, and menus. VB enables the user to control program execution. This type of program is referred to as being *event-driven*. The ability to create user-friendly event-driven programs with attractive Windows-like interfaces quickly has resulted in VB becoming one of the most popular programming languages. VB can also be thoroughly integrated with Microsoft Office to customize programs.

*Perl (Practical Extraction and Report Language)* is an interpreted language, based on C and several UNIX utilities. Perl has powerful string handling features for extracting information from text files. Perl can assemble a string and send it to the shell as a command; hence, it is often used for system administration tasks. A program in Perl is known as a *script*.

**Classification of Currently Available Languages according to The Programming Style:**

1. *Procedural/Imperative Languages:* These consist of explicit instructions that describe the step-by-step procedure to solve the problem. These are formed from collection of basic commands (assignments, input, output, etc) and control structures (selection, iteration, etc.). E.g.: C, Pascal, FORTRAN, Assembly, etc.
2. *Nonprocedural languages:* These are a programming languages that do not follow the procedural paradigm (pattern) of executing statements, subroutine calls, and control structures sequentially but instead describes a set of facts and relationships and then is queried for specific results.

3. *Functional Languages:* These are based on *lambda-calculus*, which concerns the application of functions to their arguments. Functional Language programs consist of collections of function definitions and their applications. E.g.: LISP (LISt Processing), etc.
4. *Logic Programming Languages:* Here, programs consists of collections of statements within a particular logic, such as predicate logic. E.g.: Prolog (Programming logic), etc.
5. *Object Oriented Language*: Here, programs consist of objects that interact with each other. E.g.: SIMULA (SIMUlation Language), Smalltalk, Eiffel, etc.. The object-oriented language that currently dominates the market is *C++*. It supports both object-oriented programming and non-object-oriented programming. *Java*, the language threatening C++ dominance, is a pure object-oriented language; that is, it is impossible to write a non-object-oriented program. The latest version of Java is *J2EE (Java2 Enterprise Edition)*. A relatively new language, *C# ("cee-sharp")* is Microsoft's answer to Java. It has most of the same advantages over C++ as does Java, but it is designed to work within Microsoft's *.NET environment*. The .NET environment is designed for building, deploying (organizing), and running Web-based applications. Many people referred to prior versions of Visual Basic as 'object-oriented', because VB programs used 'objects' such as command buttons, scrollbars, and others. However, it wasn't until the current version, *VB.NET*, that Visual Basic supported the concepts of inheritance and polymorphism, thus meeting the criteria for a true object-oriented language.
6. *Declarative Language:* These are collections of declarations. Many functional and Logic languages are also declarative. Here, you describe a pattern to be matched without writing the code to match the pattern.
7. *Scripting Languages:* Scripting languages are designed to perform special or limited tasks, sometimes associated with a particular application or function. E.g.: Perl (Practical extraction report language), etc.
8. *Parallel Languages*: These are collections of processes that communicate with each other. E.g.: C*, Ada, etc.

**Basic Questions when Selecting a Suitable Computer Language:**

1. How readable is the language, to humans?
2. How easy is it to write the program in this particular language?
3. How reliable is the language?
4. How much would it cost to develop using a given language?
5. How complicated is the syntax going to be?
6. Does the language have standards for greater readability? For instance Java has standards for naming, commenting and capitalization.