LANGUAGE CLASSIFICATIONS

The language used in the communication of computer instructions is known as the programming language. The computer has its own language and any communication with the computer must be in its language or translated into this language. There are two major types of computer or programming languages: low- level and high-level. The low-level languages can be further divided into machine and assembly languages.

The development of programming languages can be distinctly divided into four generations:-

1. First generation machine language
2. Second generation assembly language
3. Third generation high level language
4. Fourth generation 4-GLs

Machine Language

Machine Language or the machine code is the fundamental language of a computer and is normally written as strings of binary ls (pulse) and 0s (no pulse). The circuitry of a computer is wired in such a way that it immediately recognizes the machine language and converts it into the electrical signals needed to run the computer.

An instruction prepared in a machine language has a two-part format as shown in the figure 1.

OPCODE (Operation code)	**OPERAND (Address/Locatio n)**

Figure – 1

The first is the command or operation and it tells the computer what function to perform. Every computer has an operation code or OPCODE for each of its functions. The second part of the instruction is the OPERAND, and it tells the computer where to find or store the data or other instructions that are to be manipulated. Typical operations involve reading, adding, subtracting, writing and so on.

Now, all computers use binary digits (0s to 1s) for performing internal operations. Hence the machine language consists of strings of binary numbers and is the only one the CPU directly understands. When stored inside the computer, the symbols which make up the machine language program, are made up of 1s and 0s.

For examples, a typical program instruction to print out a number on the printer might be 10110011111010011101100.

This is definitely not a very easy language to learn, partly because it is difficult to read the understand and partly because it is written in a number system with which we are not familiar.
Since programmers are more familiar with the decimal system, most of them preferred to write the computer instructions in decimal, and leave the input device to convert these to binary. With this change the above program instruction appears as follows:

54772354

Thus the set of instruction codes, whether in binary or decimal, which is directly understood by CPU without the help of a translating program, is called machine code or machine language.

dvantages and Limitations of machine language

Programs written in machine language can be executed very fast by the computer. This is due to the fact that machine instructions are directly understood by the CPU and no translation of the program is required. But writing a program in machine language has some disadvantages which are given below:

1. *Machine dependence*: Since the internal design of a computer varies from machine to machine, the machine language is different from computer to computer. Thus a program written in machine language in one computer needs modification for its execution in another computer.
2. *Difficult to program*: A machine language programmer must have a thorough knowledge about the hardware structure of the computer.
3. *Error Prone*: for writing programs in machine language, a programmer has to remember the OPCODES and has to keep track of the storage location of data and instructions. In the process, it becomes very difficult for him to concentrate fully on the logic of the problem and as a result some errors may arise in programming.

4. *Difficult to modify*: It is very difficult to correct or modify machine language programs.

Assembly Language

The numeric machine codes (decimal or binary) are often difficult to remember and encoding is a laborious process and mistakes can be made easily. To overcome these problems, the idea of mnemonics (or memory aids) was introduced. For example, a computer may be designed to interpret the machine code 1001 (binary) or 09 (decimal) as the operation `multiply', but it is easier for the human being to remember it as MULT or

MLT. Therefore, an assembly code may consist of some users friendly mnemonics, e.g. DIV (divide), SUB (subtract), etc.

Since the computer understands only machine code instructions, a program written in assembly, language must be translated into machine language before the program is executed. **This translation is done by a special computer program known as assembler**.

Advantage of Assembly Language over Machine Language

1. *Easier to understand and use*: Assembly languages are easier to understand and use because mnemonics are used instead of numeric op-codes and suitable names are used for data.
2. *Easy to locate and correct errors*: While writing programs is assembly language, fewer errors are made and those that are made are easier to find and correct because of the use of mnemonics and symbolic field names.
3. *Easier to modify*: Assembly language programs are easier for people to modify than machine language programs. This is mainly because they are easier to understand and hence it is easier to locate, correct and modify instructions as and when desired.
4. *No worry about addresses*: The great advantage of assembly language is that it eliminates worry about address for instructions and data.

Limitations of Assembly Language:

1. *Machine dependence*: Programs written in assembly language are designed for the specific make and model of the processor being used and are therefore machine dependent.
2. *Knowledge of hardware required*: Since assembly languages are machine dependent, the programmer must be aware of a particular machine's characteristics and requirements as the program is written.

Machine and assembly codes are based on the basic design of computers and are referred to as “low-level” language.

Procedure-oriented Language

These languages consist of a set of words and symbols and one can write programs using these maintaining certain rules. These languages are oriented toward the problem to be solved or procedures of solution rather than mere computer instructions. They are also known as high-level languages. These languages enable the programmer to write instructions using English words and familiar mathematical symbols.

The most important characteristic of a high-level language is that it is machine- independent and a program written in a high-level language can be executed on computers of different make with little or no modification. The programmer does not need to know the characteristics of that machine. **Programs written in high-level language are to be translated by translator (Compiler or Interpreter)** into equivalent machine code instructions before actual implementation.

Advantages of High Level Languages

1. *Machine Independence*: High level languages are machine independent, i.e., a program written in a high-level language can be run on many different types of computers with very little or practically no effort.
2. *Easy to learn and use*: These languages are very similar to the languages normally used by us in our day-to-day life. Hence they are easy to learn and use.
3. *Fewer errors*: In these languages, since the programmer need not write all the small steps carried out by the computer, he is much less likely to make an error.
4. *Lower program preparation cost*: Writing programs in high-level languages requires less time and effort which ultimately leads to lower program preparation cost.

5. *Easier to maintai*n: Programs written in high-level languages are easier to maintain than assembly language or machine language programs. This is mainly because they are easier to understand and hence it is easier to locate, correct and modify instructions as and when desired.

Limitations of High-Level Languages

1. *Lower efficiency*: A program written in assembly language or machine language is more efficient than one written in high-level language. That is, the programs written in high-level languages take more time to run and require more main storage.
2. *Lack of flexibility*: Because the automatic features of high-level languages always occur and are not under the control of the programmer, they are less flexible than assembly languages.

High –Level versus Low-Level Languages

High-level languages are easier to learn, to understand and to write. They are easier to correct and, in general, portable (machine independent).

Low-level languages require far less space in memory, execute more quickly and permit the programmer to have more control over the internal workings of the computer. On the other hand, these languages are machine dependent and

compared to high-level languages, they are more difficult to learn and use, since the programmer requires extensive knowledge of the machine's architecture.

Some high-level languages

FORTRAN – Stands for FORmula TRANslation. This is originally developed by IBM (International Business Machine) in 1956-1957. FORTRAN was designed to solve scientific and engineering problems and

is popular among scientists and engineers.

COBOL – Stands for Common Business Oriented Language. This came around 1958. It was designed specifically for business data processing.

BASIC – Stands for Beginners All-purpose Symbolic Instruction Code. It was developed by Prof. J. Kemeny and T. Kurtz in 1964 at Dartmouth College in the United States. The language is widely used in schools and is popular for microcomputer users.

PASCAL – Named after French mathematician, Pascal. This was first introduced in 1971 by Prof. N. Wirth of the Federal Institute of Technology in Zurich Switzerland. The language may be used for both scientific & business applications.

ALGOL – Like FORTRAN, ALGOL (ALGOrithmic Language) is also one of the earliest and the most influential high-level languages, that was developed for scientific applications.

LOGO – This language was developed as part of an experiment for teaching small children. It is mainly known for its 'turtle graphics'.

What is the main difference between an interpreter and a compiler?

We usually prefer to write computer programs in languages we understand rather than in machine language, but the processor can only understand machine language. So we need a way of converting our instructions (source code) into machine language. This is done by an interpreter or a compiler.

An interpreter reads the source code one instruction or line at a time, converts this line into machine code and executes it. The machine code is

then discarded and the next line is read. The advantage of this is it's simple and you can interrupt it while it is running, change the program and either continue or start again. The disadvantage is that every line has to be translated every time it is executed, even if it is executed many times as the program runs. Because of this interpreters tend to be slow. Examples of interpreters are Basic on older home computers, and script interpreters such as JavaScript, and languages such as Lisp and Forth.

A compiler reads the whole source code and translates it into a complete machine code program to perform the required tasks which is output as a new file. This completely separates the source code from the executable file. The biggest advantage of this is that the translation is done once only and as a separate process. The program that is run is already translated into machine code so is much faster in execution. The disadvantage is that you cannot change the program without going

back to the original source code, editing that and recompiling (though for a professional software developer this is more of an advantage because it stops source code being copied). Current examples of compilers are Visual Basic, C, c++, C#, Fortran, Cobol, Ada, Pascal and so on.

You will sometimes see reference to a third type of translation program: **an assembler**. This is like a compiler, but works at a much lower level, where one source code line usually translates directly into one machine code instruction.