Introduction to Programming

An organized list of instructions that, when executed, causes the computer to behave in a predetermined manner. Without programs, computers are useless.

A program is like a recipe. It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables. The variables can represent numeric data, text, or graphical images.

There are many programming languages -- C, C++, Pascal, BASIC, FORTRAN, COBOL, and LISP are just a few. These are all high-level languages. One can also write programs in low-level languages called assembly languages, although this is more difficult. Low-level languages are closer to the language used by a computer, while high-level languages are closer to human languages.

Categories of Programming Language

Programming languages fall into two fundamental categories low and high-level languages. Low-level languages are machine-dependent; that is, they are designed to be run on a particular computer. In contrast, high-level languages (for example, COBOL and BASIC) are machine-independent and can be run on a variety of computers.

The hierarchy of programming languages contain various types of programming languages. Through the first four decades of computing, programming languages evolved in generations. The first two generations were low-level and the next two high-level generations of programming languages.

The higher-level languages do not provide us greater programming capabilities, but they do provide a more sophisticated programmer/computer interaction. In short, the higher the level of the language, the easier it is to understand and use. For example, in a fourth-generation language, you need only instruct the computer system what to do, not necessarily how to do it.

Characteristics of Programming language

The following are the characteristics of a programming language

1. **Readability**: A good high-level language will allow programs to be written in some ways that resemble a quite-English description of the underlying algorithms. If care is taken, the coding may be done in a way that is essentially self-documenting.
2. **Portability**: High-level languages, being essentially machine independent, should be able to develop portable software.
3. **Generality**: Most high-level languages allow the writing of a wide variety of programs, thus relieving the programmer of the need to become an expert in many diverse languages.
4. **Brevity**: Language should have the ability to implement the algorithm with less amount of code. Programs expressed in high-level languages are often considerably shorter than their low-level equivalents.
5. **Error checking**: Being human, a programmer is likely to make many mistakes in the development of a computer program. Many high-level languages enforce a great deal of error checking both at compile-time and at run-time.
6. **Cost:** The ultimate cost of a programming language is a function of many of its characteristics.
7. **Familiar notation**: A language should have a familiar notation, so it can be understood by most of the programmers.

8. **Quick translation**: It should admit quick translation.
9. **Efficiency**: It should permit the generation of efficient object code.
10. **Modularity**: It is desirable that programs can be developed in the language as a collection of separately compiled modules, with appropriate mechanisms for ensuring self-consistency between these modules.
11. **Widely available**: Language should be widely available and it should be possible to provide translators for all the major machines and for all the major operating systems.

Generation of Languages

1GL or first-generation language was (and still is) machine language or the level of instructions and data that the processor is given to work on (which in conventional computers is a string of 0s and 1s).

2GL or second-generation language is assembler (sometimes called "assembly") language. A typical 2GL instruction looks like this:

ADD

An assembler converts the assembler language statements into machine language.

3GL or third-generation language is a "high-level" programming language, such as PL/I, C, or Java. Java language statements look like this:

public boolean handleEvent (Event evt) { switch (evt.id) {

case Event.ACTION_EVENT: {

if ("Try me" .equald(evt.arg)) {

A compiler converts the statements of a specific high-level programming language into machine language. (In the case of Java, the output is called bytecode, which is converted into appropriate machine language by a Java virtual machine that runs as part of an operating system platform.) A 3GL language requires a considerable amount of programming knowledge.

4GL or fourth-generation language is designed to be closer to natural language than a 3GL language. Languages for accessing databases are often described as 4GLs. A 4GL language statement might look like this:

EXTRACT ALL CUSTOMERS WHERE "PREVIOUS PURCHASES" TOTAL MORE THAN $1000

5GL or fifth-generation language is programming that uses a visual or graphical development interface to create source language that is usually compiled with a 3GL or 4GL language compiler. Microsoft, Borland, IBM, and other companies make 5GL visual programming products for developing applications in Java, for example. Visual programming allows you to easily envision object-oriented programming class hierarchies and drags icons to assemble program components.

Programming paradigms

Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.

Some paradigms are concerned mainly with implications for the execution model of the language, such as allowing side effects, or whether the sequence of operations is defined by the execution model. Other paradigms are concerned mainly with the way that code is organized, such as grouping a code into units along with the state that is modified by the code. Yet others are concerned mainly with the style of syntax and grammar.

Common programming paradigms include:

- Imperative which allows side effects,
- Functional which disallows side effects,
- Declarative which does not state the order in which operations execute,
- Object-oriented which groups code together with the state the code modifies,
- Procedural which groups code into functions,
- Logic which has a style of execution model coupled to a style of syntax and grammar, and
- Symbolic programming which has a style of syntax and grammar.

Procedural oriented programming (pop):-

A program in a procedural language is a list of instruction where each statement tells the computer to do something. It focuses on procedure (function) & algorithm is needed to perform the derived computation.

When the program becomes larger, it is divided into function & each function has clearly defined purpose. Dividing the program into functions & module is one of the cornerstones of structured programming.

E.g.:- c, basic, FORTRAN.

Characteristics of Procedural oriented programming: -

- It focuses on process rather than data.
- It takes a problem as a sequence of things to be done such as reading, calculating and printing. Hence, many functions are written to solve a problem.
- A program is divided into many functions and each function has clearly defined purpose.
- Most of the functions share global data.
- Data moves openly around the system from function to function.

Drawback of Procedural oriented programming (structured programming):-

- It's emphasis on doing things. Data is given a second-class status even though data is the reason for the existence of the program.
- Since every function has complete access to the global variables, the new programmer can corrupt the data accidentally by creating function. Similarly, if new data is to be added, all the function needed to be modified to access the data.
- It is often difficult to design because the components function and data structure do not model the real world.
- For example, in designing graphical user interface, we think what functions, what data structures are needed rather than which menu, menu item and soon.
- It is difficult to create new data types. The ability to create the new data type of its own is called extensibility. Structured programming languages are not extensible.

Difference between Procedure Oriented Programming (POP) & Object-Oriented Programming(OOP)

Object-Oriented Programming	Procedure Oriented Programming	Points
In OOP, the program is divided into parts called objects.	In POP, the program is divided into small parts called functions.	Divided Into
In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.	In POP, Importance is not given to data but to functions as well as the sequence of actions to be done.	Importance
OOP follows Bottom Up approach.	POP follows Top-Down approach.	Approach
OOP has access specifiers named Public, Private, Protected, etc.	POP does not have any access specifier.	Access Specifiers
In OOP, objects can move and communicate with each other through member functions.	In POP, Data can move freely from function to function in the system.	Data Moving
OOP provides an easy way to add new data and function.	To add new data and function in POP is not so easy.	Expansion
In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.	In POP, the most function uses Global data for sharing that can be accessed freely from function to function in the system.	Data Access
OOP provides Data Hiding so provides more security.	POP does not have any proper way for hiding data so it is less secure.	Data Hiding
In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.	In POP, Overloading is not possible.	Overloading
Example of OOP are: C++, JAVA, VB.NET, C#.NET.	Example of POP are: C, VB, FORTRAN, Pascal.	Examples

The core of the pure object-oriented programming is to create an object, in code, that has certain properties and methods. While designing C++ modules, we try to see the whole world in the form of objects. For example, a car is an object which has certain properties such as color, the number of doors, and the like. It also has certain methods such as accelerate, brake, and so on.