PROGRAMMING IN C++

HISTORICAL PERSPECTIVE

The C++ programming language was created by Bjarne Stroustrup and his team at Bell Laboratories (AT&T, USA) to help implement simulation projects in an object-oriented and efficient way. The earliest versions, which were originally referred to as "C with classes," date back to 1980. As the name C++ implies, C++ was derived from the C programming language: ++ is the increment operator in C. As early as 1989 an ANSI Committee (**A**merican **N**ational **S**tandards **I**nstitute) was founded to standardize the C++ programming language. The aim was to have as manycompiler vendors and software developers as possible agree on a unified description ofthe language in order to avoid the confusion caused by a variety of dialects. In 1998 the ISO (**I**nternational **O**rganization for **S**tandardization) approved a standard for C++ (ISO/IEC 14882).

Characteristics of C++

C++ is not a purely object-oriented language but a hybrid that contains the functionalityof the C programming language. This means that you have all the features that are availablein C:

■ universally usable modular programs
■ efficient, close to the machine programming
■ portable programs for various platforms.

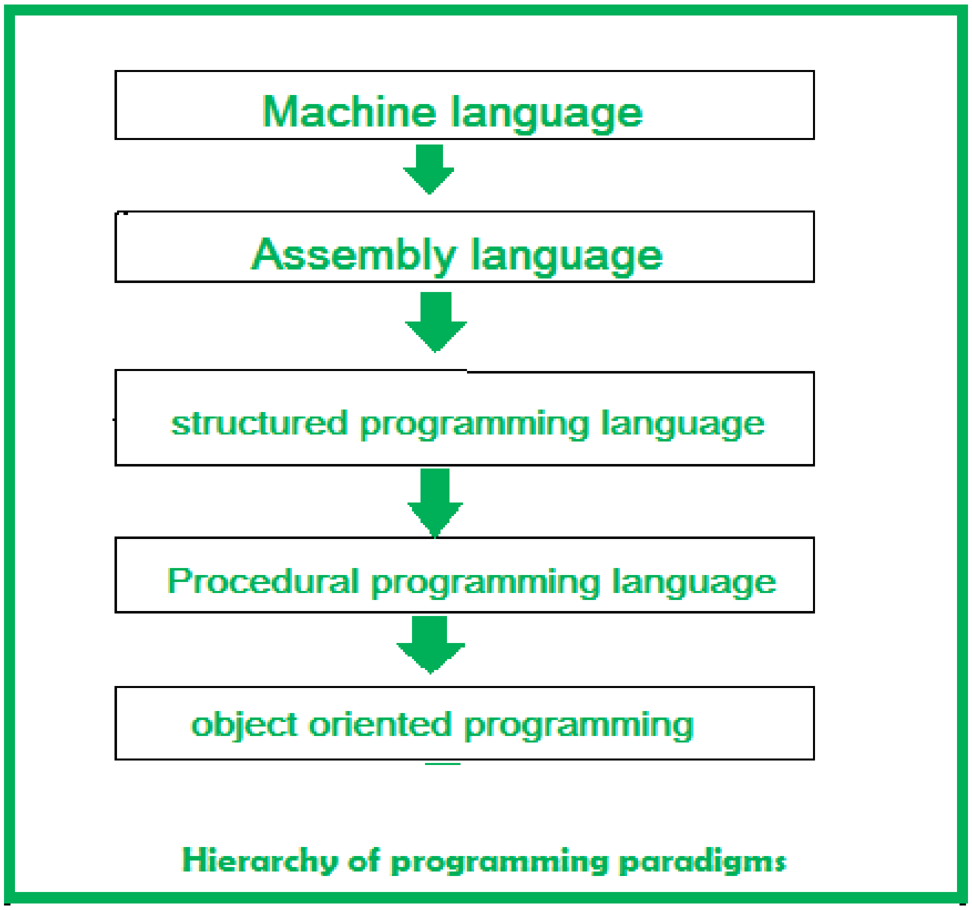
The large quantities of existing C source code can also be used in C++ programs. C++ supports the concepts of object-oriented programming (or OOP for short), which are:

■ *data abstraction,* that is, the creation of classes to describe objects
■ *data encapsulation* for controlled access to object data
■ *inheritance* by creating derived classes (including multiple derived classes)
■ *polymorphism* (Greek for multiform), that is, the implementation of instructions that can have varying effects during program execution.

Various language elements were added to C++, such as references, templates, and exception handling. Even though these elements of the language are not strictly object-

Object oriented design started right from the moment computers were invented. Programming was there, and programming approaches came into the picture. Programming is basically giving certain instructions to the computer.

At the beginning of the computing era, programming was usually limited to machine language programming. Machine language means those sets of instructions that are specific to a particular machine or processor, which are in the form of 0's and 1's. These are sequences of bits (0100110…). But it's quite difficult to write a program or develop software in machine language.


Machine language
Assembly language
structured programming language
Procedural programming language
object oriented programming
Hierarchy of programming paradigms

It's actually impossible to develop software used in today's scenarios with sequences of bits. This was the main reason programmers moved on to the next generation of programming languages, developing assembly languages, which were near enough to the English language to easily understand. These assembly languages were used in microprocessors. With the invention of the microprocessor, assembly languages flourished and ruled over the industry, but it was not enough. Again, programmers came up with something new, i.e., structured and procedural programming.

Objects:

Object-oriented programming shifts the focus of attention to the *objects*, that is, to the aspects on which the problem is centered. A program designed to maintain bank accounts would work with data such as balances, credit limits, transfers, interest calculations, and so on. An object representing an account in a program will have properties and capacities that are important for account management. OOP objects combine data (properties) and functions (capacities). A class defines a certain object type by defining both the properties and the capacities of the objects of that type. Objects communicate by sending each other "messages," which in turn activate another object's capacities.

The following three steps are required to create and translate a C++ program:

1. First, a text editor is used to save the C++ program in a text file. In other words, the *source code* is saved to a *source file*. In larger projects the programmer will normally use *modular programming*. This means that the source code will be stored in several source files that are edited and translated separately.

2. The source file is put through a *compiler* for translation. If everything works as planned, an object file made up of *machine code* is created. The object file is also referred to as a *module*.

3. Finally, the *linker* combines the object file with other modules to form an *executable file*. These further modules contain functions from standard libraries or parts of the program that have been compiled previously. It is important to use the correct file extension for the source file's *name.* Although the file extension depends on the compiler you use, the most commonly found file extensions are .cpp and .cc.
Prior to compilation, *header files*, which are also referred to as *include files*, can be copied to the source file. Header files are text files containing information needed by various source files, for example, type definitions or declarations of variables and functions.

Header files can have the file extension .h, but they may not have any file extension. The *C++ standard library* contains predefined and standardized functions that are available for any compiler.

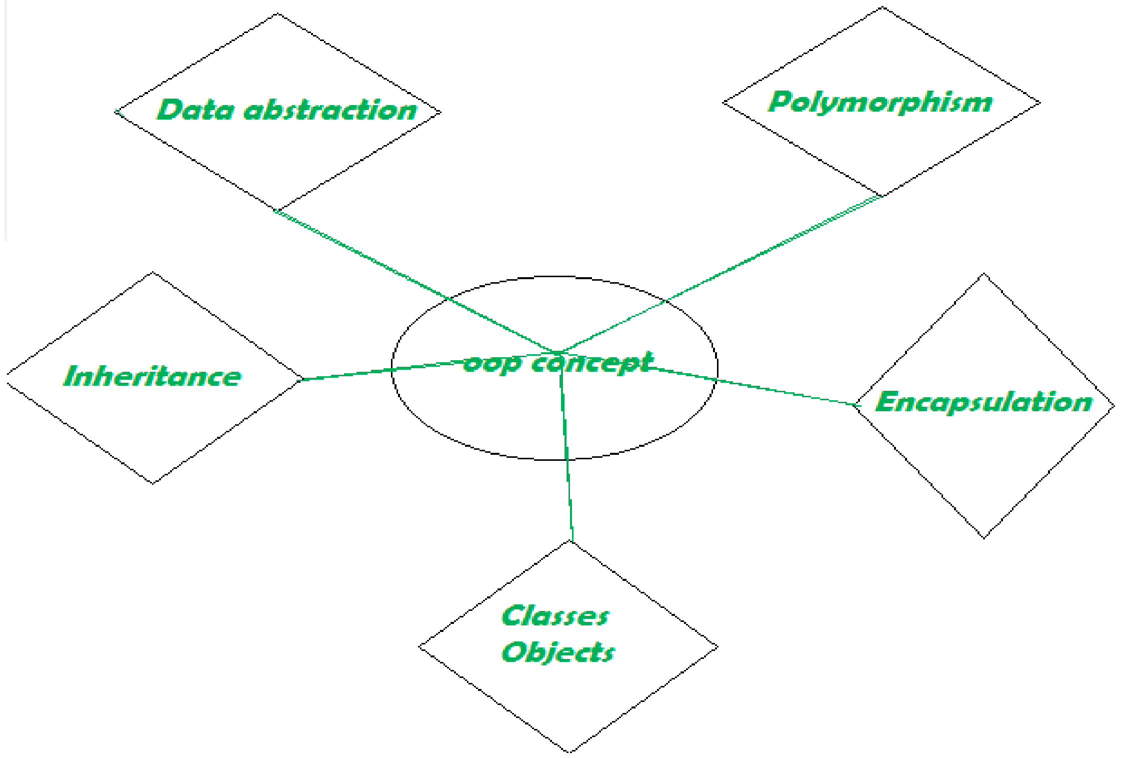
Modern compilers normally offer an *integrated software development environment*, which combines the steps mentioned previously into a single task. A graphical user interface is available for editing, compiling, linking, and running the application. Moreover, additional tools, such as a debugger, can be launched.

If the source file contains just one *syntax error*, the compiler will report an *error*. Additional error messages may be shown if the compiler attempts to continue despite having found an error. So when you are troubleshooting a program, be sure to start with the first error shown.

In addition to error messages, the compiler will also issue *warnings*. A warning does not indicate a syntax error but merely draws your attention to a possible error in the program's logic, such as the use of a non-initialized variable.

Structured Programming :

The basic principle of the structured programming approach is to divide a program into functions and modules. The use of modules and functions makes the program more understandable and readable.
It helps to write cleaner code and to maintain control over the functions and modules. This approach gives importance to functions rather than data.
It focuses on the development of large software applications, for example, C was used for modern operating system development. The programming languages: PASCAL (introduced by Niklaus Wirth) and C (introduced by Dennis Ritchie) follow this approach.



Procedural Programming Approach :

This approach is also known as the top-down approach. In this approach, a program is divided into functions that perform specific tasks. This approach is mainly

used for medium-sized applications. Data is global, and all the functions can access global data.
The basic drawback of the procedural programming approach is that data is not secured because data is global and can be accessed by any function. Program control flow is achieved through function calls and goto statements. The programming languages: FORTRAN (developed by IBM) and COBOL (developed by Dr Grace Murray Hopper) follow this approach.

These programming constructs were developed in the late 1970s and 1980s. There were still some issues with these languages, though they fulfilled the criteria of well-structured programs, software etc. They were not as structured as the requirements were at that time. They seem to be over-generalised and don't correlate with real-time applications. To solve such kinds of problems, OOP, an object-oriented approach was developed as a solution.

The Object-Oriented Programming (OOP) Approach :

The OOP concept was basically designed to overcome the drawback of the above programming methodologies, which were not so close to real-world applications. The demand was increased, but still, conventional methods were used. With the emergence of OOP, all conventional methods were thrown out the window.
OOP methodology not only solved the modern-era problem, but also, it brought a revolution in the programming methodology field.

Object-oriented programming (OOP) is nothing but that which allows the writing of programs with the help of certain classes and real-time objects. We can say that this approach is very close to the real-world and its applications because the state and behaviour of these classes and objects are almost the same as real-world objects.
Basic Concepts of Object Oriented Programming using C++
Object oriented programming – As the name suggests uses objects in programming.

Object oriented programming aims to implement real world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operates on them so that no other part of code can access this data except that function.
Let us learn about different characteristics of an Object Oriented Programming language:

Object: Objects are basic run-time entities in an object oriented system, objects are instances of a class these are defined user defined data types.
ex:

```
        class person
{
   char name[20];
```

```
    int id;
public:
    void getdetails(){}
};

int main()
{
   person p1; //p1 is a object
}
```

Object take up space in memory and have an associated address like a record in pascal or structure or union in C.
When a program is executed the objects interact by sending messages to one another. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each others data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

Class: Class is a blueprint of data and functions or methods. Class does not take any space.
Syntax for class:

```
class class_name
{
  private:
     //data members and member functions
declarations
  public:
     //data members and member functions
declarations
  protected:
     //data members and member functions
declarations
};
```

Class is a user defined data type like structures and unions in C.
By default class variables are private but in case of structure it is public. in above example person is a class.

Inheritance: inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. Inheritance provides re usability. This means that we can add additional features to an existing class without modifying it.

Polymorphism:

Polymorphism means ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.C++ supports operator overloading and function overloading.
The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.
Function overloading is using a single function name to perform different types of tasks. Polymorphism is extensively used in implementing inheritance.

Some important points to know about OOP:

1. OOP treats data as a critical element.
2. Emphasis is on data rather than procedure.
3. Decomposition of the problem into simpler modules.
4. Doesn't allow data to freely flow in the entire system, ie localized control flow.
5. Data is protected from external functions.

Advantages of OOP

Object-oriented programming offers several major advantages to software development:
■ **reduced susceptibility to errors**: an object controls access to its own data. More specifically, an object can reject erroneous access attempts
■ **easy re-use**: objects maintain themselves and can therefore be used as building blocks for other programs
■ **low maintenance requirement**: an object type can modify its own internal data representation without requiring changes to the application.

- It models the real world very well.
- With OOP, programs are easy to understand and maintain.
- OOP offers code reusability. Already created classes can be reused without having to write them again.
- OOP facilitates the quick development of programs where parallel development of classes is possible.
- With OOP, programs are easier to test, manage and debug.
- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs form the standard working modules that communicate with one another, rather than having to start writing the code form scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programmer to build secure programs that cannot be invaded by code in other parts of the program.