## Procedure Oriented Programming

Conventional programming using high level languages such as COBOL, FORTRAN and C is commonly known as procedure oriented programming. In the procedure oriented approach the problem is viewed as a sequence of things to done, such as reading, calculating and printing. A number of function are written to accomplish these tasks. The primary focus is on functions.

Procedure oriented programming basically consists of writing a list of instruction for the computer to follow and organizing these instruction into groups known as functions. We normally use flowchart to organize these actions and represent the flow of control from one action to another.

In a multi-function program, may important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data.

Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we should also revised all function that access the data.

## Characteristics of procedure oriented programming are

- Emphasis is on doing things (algorithms)
- Large program are divided into smaller program known as function
- Most of the function share global data
- Data move openly around the system from function to function
- Functions transform data from one form to another
- Employs top-down approach in program design.

## Object oriented programming

The major motivating factor in the invention of object-oriented approach is to salvage some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modification from outside function. OOP allows us to decompose a problem into a number of entities called objects and then build data and functions around these entities.

The data of an object can be accessed only by the functions associated with that objects. However, functions of one object can access the functions of other objects.

## Characteristics of procedure oriented programming are

- Emphasis is on data rather than procedure
- Large program are divided into what are known as object
- Data structures are designed such that they characterize the objects
- Functions that operate on the data of an object are tied together in the data structure
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program design

# Definition of Object Oriented Program

Object-oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

# Basic concepts of object-oriented programming

'Object-oriented' remains a term which is interpreted differently by different people. It is therefore necessary to understand some of the concepts used extensively in object-oriented programming. Following are the general concepts:

1. **Objects**
2. **Data abstraction**
3. **Inheritance**
4. **Dynamic binding**
5. **Classes**
6. **Data encapsulation**
7. **Polymorphism**
8. **Message passing**

# Objects:

Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or ay item that the program must handle. They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in terms of objects and the nature o communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have an associated address like a record in Pascal or a structure 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 other's data or code. It is sufficient to know the type of message accepted and the type of response returned by the objects.

# Classes:

The entire set of data and code of an object can be made a user-defined data type with the help of a class. In fact, objects are variable of type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus collection of objects of similar type. Classes are user-defined data types and behave like the built-in types of a programming language.

# Data Abstraction and Encapsulation

The wrapping up of data and functions into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking features of a class. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called data hiding.

A abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and functions to operate on these attributes. They encapsulate all the essential properties of the object that are to be created. Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).

## Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It support the concept of hierarchical classification.

In OOP, the concept of inheritance provides the idea of reusability. This mean that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

## Polymorphism

Polymorphism is another important OOP concepts. Polymorphism means the ability to take more than one form. Polymorphism plays an important role in allowing objects having different internal structure to share the same external interface. This means that a general class of operation may be access the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

## Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depend on the dynamic type of that reference.

## Message Communication

An object-oriented program consists of a set of objects that communication with each other. The process of programming in an object-oriented language therefore involves the following basic steps.

- Creating classes that define objects and their behavior
- Creating objects from class definitions
- Establishing communication among objects

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. A message for an object is a request for execution of a procedure, and therefore will invoke a function(procedure) in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function(message) and the information to be sent.

## Benefits of OOP

OOP offers several benefits to both the program designer and the user. The principal advantages are

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build program from the standard working modules that communicate with one another, rather than having o start writing the code from scratch.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those objects in the program.
- It is easy to partition the work in a project based on objects.
- The data-center design approach enables us to capture more details of a model in implementable form.
- Object oriented system can be easily upgraded from small to large system.
- Software complexity can be easily managed.

# What is C++?

C++ is an object-oriented programming language. Initially named 'C with classes', C++ was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA. In the early eighties, Stroustrup, and admirer of Simula67 and a strong supporter of C, wanted to combine the best of both languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++. Therefore class was a major addition to the original C language, Stroustrup called a new language 'C' with classes'. However, later in 1983, the name was changed to C++. The Idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented (incremented) version of C.

C++ is a superset of C. most of what we already know about C applies to C++ also. Therefore almost all C Program are also C++ programs. However there are a few minor difference that will prevent a C program to run under C++ compiler.

The three most important facilities that C++ adds on C are classes, function overloading, and operator overloading. These features enable us to create abstract data types, inherit properties from existing data types sand support polymorphism, thsu making C++ truly object-oriented langage.

# Program Features

Like C, the C++ program is a collection of function. The program contains only one function main(). As usual, execution begins at main(). Every C++ program must have a main(). C++ is free-form language. With a few exceptions, the compiler ignores return and white spaces. Like C, the C++ statements terminate with semicolons.

# Comments

C++ introduces a new comment symbol // (double slash). Comments start with a double slash symbol and terminate at the end of the line. A comment may start anywhere in the line and whatever follows till end of the line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment.

# Example:

```
//This is an example of
//C++ program to illustrate
```

The C comment symbol /*, */ are still valid and are more suitable for multiline comments. The following comment is allowed.

```
/* This is an example of
C++ program to illustrate
Some of its features*/
```

We can use either or both styles in our programs.

# Output Operator

```
cout<<"C++ is better than C";
```

This statement causes the string in quotation marks to be displayed on the screen. This statement introduces two new C++ features, **cout** and **<<**. The identifier **cout** (pronounced as 'C out') is a predefined object that represents the standard output stream in C++. Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices

The operator **<<** is called the **insertion or put** to operator. It inserts (or sends) the contents of the variable on its right to the object on its left.

It is important to note that we can still use **printf()** for displaying an output. C++ accepts this notation. However, we will use **cout <<** to maintain the spirit of C++.

## Input Operator

cin >> number1;

This is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier **cin** (pronounced 'C in') is a predefined object in C++ that corresponds to the standard input stream. Here, this stream represent the keyboard.

The operator **>>** is known as **extraction or get from operator**. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right. This corresponds to the familiar **scanf()** operations. Like **<<**, the operator **>>** can also be overloaded.

## Cascading of I/O Operators

The multiple use of << in one statement is called cascading. When cascading an output operator, we should ensure necessary blank spaces between different items. Using the cascading technique, the two statement as follows

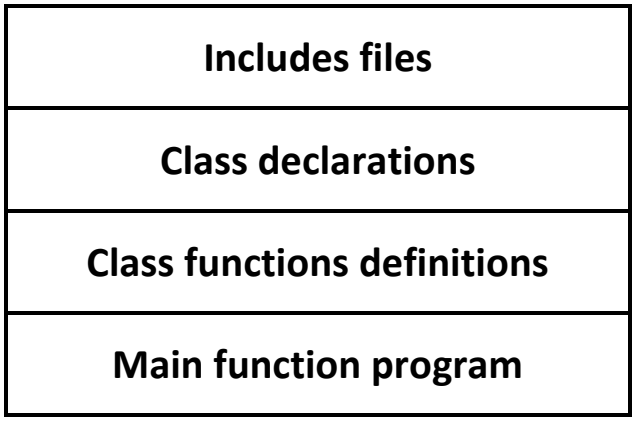
cout << "Sum=" << sum << "\n" << "Average = " << average << "\n";

## Example with class

```
#include<iostream.h>
class person
{
char name[3];
int age;
public:
void getdata(void);
void putdata(void);
};
void person :: getdata(void)
{
cout << "\nEnter Name:";
cin >> name;
cout << "Enter Age:";
cin >> age;
}
void person :: putdata(void)
{
cout << "\nName:" << name;
cout << "\nAge :" << age;
}
void main()
{
person p;
p.getdata();
p.putdata();
}
```

## Structure of C++ Program

Typically C++ program would contain four sections.




TOKENS, EXPRESSIONS AND CONTROL STRUCTURES

## TOKENS

The smallest individual units in a program are known as tokens. C++ has the following token
Keywords
Identifiers
Constants
Strings
Operator
A C++ program is written using these tokens, white spaces, and the syntax of the language. Most of the C++ tokens are basically similar to the C tokens with the exception of some additions and minor modification.

## Keywords

The keywords implement specific C++ language features. They are explicitly reserved identifiers and cannot be used as names for the program variables or other user-defined program elements.

## Identifiers

Identifiers refer to the names of variables, functions, arrays, classes, etc. created by the programmer. They are the fundamental requirement of any language. Each language has its own rules for naming these identifiers. The following rules are common to both C & C++.

- Only alphabetic characters, digits and underscores are permitted
- The name cannot start with a digit.
- Uppercase and lowercase letters are distinct.
- A declared keyword cannot be used as a variable name

The major difference between C and C++ is the limit on the length of a name. While ANSI C recognizes only the first 32 characters in a name, C++ places no limit on its length and therefore all the characters in a name are significant.

## Basic data types

Both C and C++ compilers support all the built-in (also known as basic or fundamental) data types. With the exception of void, the basic data types may have several modifiers preceding them to serve the needs of various situations. The modifiers signed, unsigned, long and short may be applied to character and integer basic data types.

The type void was introduced in ANSI C. Two normal uses of void are (1) to specify the return type of a function when it is not returning any value, and (2) to indicate an empty argument list to a function. Example void function1(void);

## USER-DEFINED DATA TYPES

We have used user-defined data types such as **struct** in C. while these data types are legal in C++, some features have been added to make them suitable for object-oriented programming. C++ also permits us to define another user-defined data type known as **Class** which can be used, just like any other basic data type, to declare variables. The class variables are known as objects, which are the central focus of object-oriented programming.

## Enumerated Data Type

An enumerated data type is another user-defined type which provides away for attaching names to numbers, thereby increasing comprehensibility of the code. The enum keyword (from C) automatically enumerates a list of words by assigning them values 0, 1, 2, 3 and so on. This facility provides an alternative means for creating symbolic constants.

Syntax

Enum <alias name>

{

Elements of enum;

}

Examples

Enum shape {Circle, square, triangle};

Enum colour {red, blue, green, yellow};

Enum position {off, on};

The enumerated data types differ slightly in C++ when compared with those in ANSI C. in C++, the tag names shape, colour, and position become new type names. That means we can declare new variable using these tag names. Example

Shape ellipse;

Colour background;

In C++ each enumerated data type retains its own weparate type. This means that C++ does not permit an int value to be automatically converted to an enum value.

Colour background = blue; // allowed

Colour background = 7;  //Error in C++

Colour background = (colour) 7; // ok

By default the enumerators are assigned integer values starting with 0 for the first enumerator, 1 for second and so on. We can over-ride the default by explicitly assigning integer values to the enumerator. For example

Enum colour {red, blue= 4, green =8};

Enum colour {red=5, blue, green};

## Derived Data Types

## Array

The application of arrays in C++ is similar to that in C. the only exception is the way character arrays are initialized. When initializing a character array in ANSI C, the compiler will allow us to declare the array size as the exact length of the string constant. For example

Char string[3]=”xyz”;

## Functions

Functions have undergone major changes in C++. While some of these changes are simple, others require a new way of thinking when organizing our programs. Many of these modification and improvements were driven by the requirement of the object-oriented concept of C++. Some of these were introduced to make the C++ program more reliable and readable.

## Pointers

Pointers are declared and initialized as in C. C++ adds the concept of constant pointer and pointer to a constant

```
Char *const ptr1=”Good”;
```

We cannot modify the address that ptr1 is initialized to

```
Int const *ptr2=&m;
```

## Symbolic Constant

There are two ways of creating symbolic constants in C++

1) Using the qualifier const
2) Defining a set of integer constant using enum keyword

In both C and C++, any value declared as const cannot be modified by the program in any way. However, there are some difference in implementation. In C++ we can use const in a constant expression, such as

```
Const int size=10;
Char name[size];
```

## Declaration of variables

In C, all variables must be declared before they are used in executable statements. This is true with C++ as well. However, there is a significant difference between C and C++ with regard to the place of their declaration in the program. C requires all the variable to be defined at the beginning of a scope. When we read a C program, we usually come across a group of variable declaration at the beginning of each scope level. Their actual use appears elsewhere in the scope, sometimes far away from the place of declaration. Before using a variable, we should go back to the beginning of the program to se whether it has been declared and fi so of what type it is.

C++ allows the declaration of a variable anywhere in the scope. This means that a variable can be declared right at the place of tis first use. This makes the program much easier to write and reduces the errors that may be caused by having so scan back and forth. It also makes the program easier to understand because the variables are declared in the context of their use.

```
Void main()
{
Float x;
Float sum =0;
For(int i=1;i<5;i++)
{
Cin>>x;
Sum=sum+x;
}
Float average;
Average = sum/I;
Cout<<average;
}
```

## Dynamic initialization of variables

One additional features of C++ is that it permits initialization of the variables at run time. This is referred to as dynamic initialization. Remember that, in C, a variable must be initialized using a

constant expression and the C compiler would fix the initialization code at the time of compilation. However, in C++a variable an be initialized at run time using expression at the place of declaration.
Example
Float area=3.14 * r * r;
Float average = sum/5;
This means that both the declaration and initialization of a variable can be done simultaneously at the place where the variable is used for the first time.
Dynamic initialization is extensively used in object-oriented programming. We can create exactly the type of object needed using information that is known only at the run time.

## Reference variables

C++ introduces a new kind of variable known as the reference variable. A reference variable provides an alias (alternative name) for a previously defined variable). For example. If we make the variable sum as a reference to the variable total, then sum and total can be used interchangeably to represent that variable.
Syntax
Data-type &reference-name=variable name;
Example
Float total=100;
Float &sum=total;
Total is a float type variable that has already been declared. Sum is the alternative name declared to represent the variable total. Both the variables refer to the same data object in the memory. Now the statement
Cout <<total
Cout <<sum;
Both print the value 100. The statement
Total=total+10;
Will change the value of both total and sum to 110

## FUNCTION IN C++

Functions play an important role in C program development. Dividing a program into functions is one of the major principles of top-down, structured programming. Another advantage of using functions is that it is possible to reduce the size of a program by calling and using them at different places in the program. C++ is no exceptions. Functions continue to be the building blocks of C++ program. In fact, C++ has added many new features to functions to make them more reliable and flexible. Like C++ operators, a C++ function can be overloaded to make it perform different tasks depending on the arguments passed to it.

## Function prototyping

Function prototyping is one of the major improvements added to C++ functions. The prototype describes the function interface to the compiler by giving details such as the number and type of arguments and the type of return values. With function prototyping, a template is always used when declaring and defining a functions. When a function is called, the compiler uses the template to ensure that proper arguments are passed, and the return value is treated correctly. Any violation in matching the arguments or the return types will be caught by the compiler at the time of compilation itself. These checks and controls did not exist in the conventional C functions.
Functions prototype is a declaration statement in the calling program and is of the following form:

*Type function-name (argument-list);*

The argument-list contains the types and names of arguments that must be passed to the function.
Example:

**float volume(int x, float y, float z);**

Note that each argument variable must be declared independently inside the parenthesis. That is a combined declaration like

**float volume(int x, float y, z);**

Isillegal.

In a function declaration, the names of the argument are dummy variable and therefore, they are optional that is in the form

**flaot volume(int, float, float);**

## CALL BY REFERENCE

In traditional C, a function call passes arguments by value. The called function creates a new set of variables and copies the values of arguments into them. The function does not have access to the actual variables in the calling program and can only work on the copies of values. This mechanism is fine if the function does not need to alter the values of the original variables in the calling program. But, there may arise situations where we would like to change the values of variables in the calling program.

Provision of the reference variables in C++ permits us to pass parameters to the functions by reference. When we pass arguments by reference, the 'formal' arguments in the called function become aliases to the 'actual' arguments in the calling function. This means that when the function is working with its own arguments, it is actually working on the original data.

Example

```
void swap(int &a, int &b)
{
int t=a;
a=b;
b=t;
}
```

## DEFAULT ARGUMENTS

C++ allows us to call a function without specifying all its arguments. In such cases, the function assigns a default value to the parameter which does not have a matching argument in the function call. Default values are specified when the function is declared. The compiler looks at the prototype to see how many arguments a functions uses and alerts the program for possible default value.

Example

**float amout(float principal, int period, float rate=0.15);**

the default value is specified in a manner syntactically similar to a variable initialization. The above program prototype declares a default value of 0.15 to the argument rate. A subsequent functions call like.

**Value=amount(5000,7);**

Passes the value of 5000 to principal and 7 to period and then lets the function use default value of 0.15 for rate. The call

**Value=amount(5000,5,0.12);**

**Passes an explicit value of 0.12 to rate.**

A default argument is checked for type at the time of declaration and evaluated at the time of call. One important point to note is that only the trailing arguments can have default values. That is, we must add defaults from right to left. We cannot provide a default value to particular argument in the middle of an argument list.

## FUNCTION OVERLOADING

Overloading refers to the use of the same thing for different purposes. C++ also permits overloading of functions. This means that we can use the same function name to create functions that performs a variety of different tasks. This is known as function polymorphism in OOP).

Using the concept of function overloading, we can design a family of functions with one function name but with different argument lists. The function would perform different operations depending on the argument list in the function call. The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.

```
//Declarations
Int add(int a, int b);
Int add(int a, int b, int c);
Double add(double x, double y);
Double add(int p, double q);
//Function calls
Cout<<add(5,10);
Cout<<add(15,10.0);
Cout<<add(12.5,7.5);
Cout<<add(5.10);
```