UNIT 1:

Principles of Object Oriented Programming (OOP)

Basic Concepts of Object Oriented Programming

Object oriented programming is a type of programming which uses objects and classes its functioning. The object oriented programming is based on real world entities like inheritance, polymorphism, data hiding, etc. It aims at binding together data and function work on these data sets into a single entity to restrict their usage.

Some basic concepts of object oriented programming are −

- CLASS
- OBJECTS
- ENCAPSULATION
- POLYMORPHISM
- INHERITANCE
- ABSTRACTION

Class − A class is a data-type that has its own members i.e. data members and member functions. It is the blueprint for an object in object oriented programming language. It is the basic building block of object oriented programming in c++. The members of a class are accessed in programming language by creating an instance of the class.

Some important properties of class are −

- **Class** is a user-defined data-type.
- A class contains members like data members and member functions.
- **Data members** are variables of the class.
- **Member functions** are the methods that are used to manipulate data members.
- Data members define the properties of the class whereas the member functions define the behaviour of the class.

A class can have multiple objects which have properties and behaviour that in common for all of them.

Syntax

```
class class_name {
  data_type data_name;
  return_type method_name(parameters);
}
```

Object − An object is an instance of a class. It is an entity with characteristics and behaviour that are used in the object oriented programming. An object is the entity that is created to allocate memory. A class when defined does not have memory chunk itself which will be allocated as soon as objects are created.

Syntax

```
class_name object_name;
```

Example

```
#include<iostream>
using namespace std;
class calculator {
  int number1;
  int number2;
  char symbol;
  public :
  void add() {
    cout<<"The sum is "<<number1 + number2 ;
  }
  void subtract() {
    cout<<"The subtraction is "<<number1 - number2 ;
  }
  void multiply() {
    cout<<"The multiplication is "<<number1 * number2 ;
  }
  void divide() {
    cout<<"The division is "<<number1 / number2 ;
  }
  calculator (int a , int b , char sym) {
    number1 = a;
    number2 = b;
    symbol = sym;
    switch(symbol){
      case '+' : add();
        break;
      case '-' : add();
        break;
      case '*' : add();
        break;
      case '/' : add();
        break;
      default : cout<<"Wrong operator";
    }
  }
};
int main() {
  calculator c1(12 , 34 , '+');
}
```

Output

```
The sum is 46
```

Encapsulation In object oriented programming, encapsulation is the concept of wrapping together of data and information in a single unit. A formale defination of encapsulation would be: encapsulation is binding togather the data and related function that can manipulate the data.

Let’s understand the topic with an easy real life example,

In our colleges, we have departments for each course like computer science, information tech. , electronics, etc. each of these departments have their own students and subjects that are kept track of and being taught. let's think of each department as a class that encapsulates the data about students of that department and the subjects that are to be taught. Also a department has some fixed rules and guidelines that are to be followed by the students that course like timings, methods used while learning, etc. this is encapsulation in real life, there are data and there are ways to manipulate data.

Due to the concept of encapsulation in object oriented programming another very important concept is possible, it is data abstraction or Data Hiding. it is possible as encapsulating hides the data at show only the information that is required to be displayed.

Polymorphism The name defines polymorphism is multiple forms. which means polymorphism is the ability of object oriented programming to do some work using multiple forms. The behaviour of the method is dependent on the type or the situation in which the method is called.

Let’s take a real life example, A person can have more than one behaviour depending upon the situation. like a woman a mother, manager and a daughterAnd this define her behaviour. This is from where the concept of polymorphism came from.

In c++ programming language, polymorphism is achieved using two ways. They are operator overloading and function overloading.

Operator overloading In operator overloading and operator can have multiple behaviour in different instances of usage.

Function overloading Functions with the same name that can do multiple types based on some condition.

Inheritance it is the capability of a class to inherit or derive properties or characteristics other class. it is very important and object oriented program as it allows reusability i.e. using a method defined in another class by using inheritance. The class that derives properties from other class is known as child class or subclass and the class from which the properties are inherited is base class or parent class.

C plus plus programming language supports the following types of inheritance

- single inheritance
- multiple inheritance
- multi level inheritance
- Hierarchical inheritance
- hybrid inheritance

Abstraction Data abstraction or Data Hiding is the concept of hiding data and showing only relevant data to the final user. It is also an important part object oriented programing.

let's take real life example to understand concept better, when we ride a bike we only know that pressing the brake will stop the bike and rotating the throttle will accelerate but you don't know how it works and it is also not think we should know that's why this is done from the same as a concept data abstraction.

In C plus plus programming language write two ways using which we can accomplish data abstraction −

- using class
- using header file

Differences between Procedural and Object Oriented Programming

Both Procedural Programming and Object Oriented Programming are high-level languages in programming world and are widely used in the development of applications. On the basis of nature of developing the code, both languages have different approaches on basis of which both are differentiate from each other.

In this article, we will discuss the important differences between procedural oriented programming and object oriented programming. But before going into the differences, let's start with some basics.

What is Procedural Programming?

Procedural Programming is a programming language that follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. In procedural oriented programming, each step is executed in a systematic manner so that the computer can understand what to do.

The programming model of the procedural oriented programming is derived from structural programming. The concept followed in the procedural oriented programming is called the "procedure". These procedures consist several computational steps that are carried out during the execution of a program. Examples of procedural oriented programming language include – C, Pascal, ALGOL, COBOL, BASIC, etc.

What is Object Oriented Programming?

Object-oriented Programming is a programming language that uses classes and objects to create models based on the real world environment. These objects contain data in the form of attributes and program codes in the form of methods or functions. In OOP, the computer programs are designed by using the concept of objects that can interact with the real world entities.

We have several types of object oriented programming languages, but the most popular is one among all is class-based language. In the class-based OOP languages, the objects are the instances of the classes that determine their types. Examples of some object oriented programming languages are – Jave, C++, C#, Python, PHP, Swift, etc.

The following table highlights the major differences between Procedural Programming and Object Oriented Programming –

Parameter	Object Oriented Programming	Procedural Programming
Definition	Object-oriented Programming is a programming language that uses classes and objects to create models based on the real world environment. In OOPs, it makes it easy to maintain and modify existing code as new objects are created inheriting characteristics from existing ones.	Procedural Programming is a programming language that follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. Each step is carried out in order in a systematic manner so that a computer can understand what to do.
Approach	In OOPs concept of objects and classes is introduced and hence the program is divided into small chunks called objects which are instances of classes.	In procedural programming, the main program is divided into small parts based on the functions and is treated as separate program for individual smaller program.
Access modifiers	In OOPs access modifiers are introduced namely as Private, Public, and Protected.	No such modifiers are introduced in procedural programming.
Security	Due to abstraction in OOPs data hiding is possible and hence it is more secure than POP.	Procedural programming is less secure as compare to OOPs.
Complexity	OOPs due to modularity in its programs is less complex and hence new data objects can be created easily from existing objects making object-oriented programs easy to modify	There is no simple process to add data in procedural programming, at least not without revising the whole program.
Program division	OOP divides a program into small parts and these parts are referred to as objects.	Procedural programming divides a program into small programs and each small program is referred to as a function.

Importance	OOP gives importance to data rather than functions or procedures.	Procedural programming does not give importance to data. In POP, functions along with sequence of actions are followed.
Inheritance	OOP provides inheritance in three modes i.e. protected, private, and public	Procedural programming does not provide any inheritance.
Examples	C++, C#, Java, Python, etc. are the examples of OOP languages.	C, BASIC, COBOL, Pascal, etc. are the examples POP languages.

Benefits of OOPS

- We can build the programs from standard working modules that communicate with one another, rather than having to start writing the code from scratch which leads to saving of development time and higher productivity,
- OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time).
- The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.
- OOP systems can be easily upgraded from small to large systems.
- It is possible that multiple instances of objects co-exist without any interference,
- It is very easy to partition the work in a project based on objects.
- It is possible to map the objects in problem domain to those in the program.
- The principle of data hiding helps the programmer to build secure programs which cannot be invaded by the code in other parts of the program.
- By using inheritance, we can eliminate redundant code and extend the use of existing classes.
- Message passing techniques is used for communication between objects which makes the interface descriptions with external systems much simpler.
- The data-centered design approach enables us to capture more details of model in an implementable form.

Operator Overloading

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.
In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a

class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc.

Example:

int a;

float b,sum;

sum = a + b;

Here, variables “a” and “b” are of types “int” and “float”, which are built-in data types. Hence the addition operator ‘+’ can easily add the contents of “a” and “b”. This is because the addition operator “+” is predefined to add variables of built-in data type only.

Example:

- C++

```
// C++ Program to Demonstrate the

// working/Logic behind Operator

// Overloading

class A {

    statements;

};



int main()

{

    A a1, a2, a3;
    a3 = a1 + a2;



    return 0;

}
```

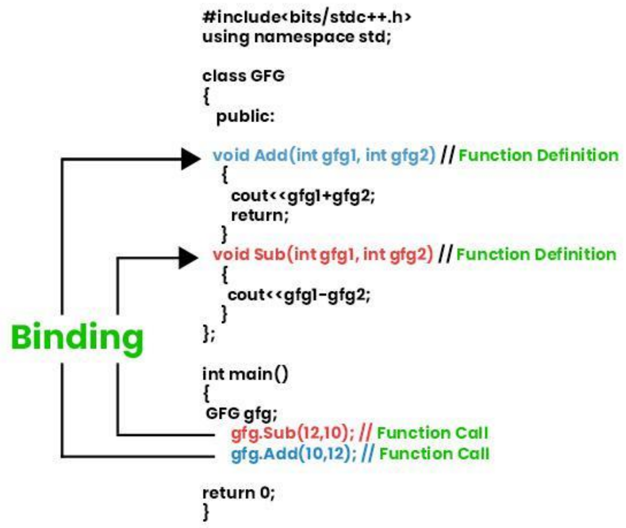
In this example, we have 3 variables “a1”, “a2” and “a3” of type “class A”. Here we are trying to add two objects “a1” and “a2”, which are of user-defined type i.e. of type “class A” using the “+” operator. This is not allowed, because the addition operator “+” is predefined to operate only on built-in data types. But here, “class A” is a user-defined type, so the compiler generates an error. This is where the concept of “Operator overloading” comes in.

Now, if the user wants to make the operator “+” add two class objects, the user has to redefine the meaning of the “+” operator such that it adds two class objects. This is done by using the concept of “Operator overloading”. So the main idea behind “Operator overloading” is to use C++ operators with class variables or class objects. Redefining the meaning of operators really does not change their original meaning; instead, they have been given additional meaning along with their existing ones.

Dynamic Binding

Dynamic binding in C++ is a practice of connecting the function calls with the function definitions by avoiding the issues with static binding, which occurred at build time. Because dynamic binding is flexible, it avoids the drawbacks of static binding, which connected the function call and definition at build time.

In simple terms, Dynamic binding is the connection between the function declaration and the function call.



```
#include<bits/stdc++.h>
using namespace std;

class GFG
{
  public:
   void Add(int gfg1, int gfg2) //Function Definition
    {
     cout<<gfg1+gfg2;
     return;
    }
   void Sub(int gfg1, int gfg2) //Function Definition
    {
     cout<<gfg1-gfg2;
    }
};

int main()
{
 GFG gfg;
     gfg.Sub(12,10); // Function Call
     gfg.Add(10,12); // Function Call

return 0;
}
```

Dynamic Binding in C++

So, choosing a certain function to run up until runtime is what is meant by the term **Dynamic binding**. A function will be invoked depending on the kind of item.

Usage of Dynamic Binding

It is also possible to use dynamic binding with a single function name to handle multiple objects. Debugging the code and errors is also made easier and complexity is reduced with the help of Dynamic Binding.

BASIC PROGRAM CONSTRUCTION IN C++

```
#include<iostream>

using namespace std;

int main()

{

cout<<" Hello World";

return 0;

}
```

OUTPUT:

```
Hello World
```

EXPLANATION OF ABOVE PROGRAM:

Despite its small size, this program demonstrates a great deal about the,construction of C++ programs. Let's examine it in detail.

FUNCTION:

Functions are one of the fundamental building blocks of C++. The program consists of almost entirely a single function called **main()**.

FUNCTION NAME:

The parentheses following the word main are the distinguishing feature of a function. Without the parentheses the compiler would think that main refers to a variable or to some other program element.

The word **int** preceding the function name indicates that this particular function has a return value of type int.

BRACES AND FUNCTION BODY:

The body of a function is surrounded by braces (sometimes called curly brackets). These

braces play the same role as the BEGIN and END keywords in some other languages. Every function must use this pair of braces around the function body.

In this example there are only two statements in the function body:

- The line starting with cout.
- And the line starting with return.

However, a function body can consist of many statements.

ALWAYS START WITH MAIN():

When you run a C++ program, the first statement executed will be at the beginning of a function called main(). The program may consist of many functions, classes, and other program elements, but on startup,control always goes to main(). If there is no function called main() in your program, an error will be reported when you run the program.

PROGRAM STATEMENT:

The program statement is the fundamental unit of C++ programming. There are two statements in the above program, the line:

1) cout << “Hello World”;

2) return statement:

return 0;

The first statement tells the computer to display the quoted phrase.The last statement in the function body is return 0;. This tells main() to return the value 0 to
whoever called it, in this case the operating system or compiler.

OUTPUT USING COUT:

As you have seen, the statement:

cout << “Hello World”;

causes the phrase in quotation marks to be displayed on the screen

DIRECTIVES:

The two lines that begin the FIRST program are directives.

- The first is a preprocessor directive,
- And the second is a using directive.

They’re not part of the basic C++ language, but they’re necessary anyway.

PREPROCESSOR DIRECTIVES:

The first line of the FIRST program

#include <iostream>

might look like a program statement, but it’s not. It isn’t part of a function body and doesn’t end with a semicolon, as program statements must. Instead, it starts with a number sign (#).

It’s called a preprocessor directive. The program statements are instructions to the computer to do something, such as: adding two numbers or printing a sentenc. A preprocessor directive, on the other hand, is an instruction to the compiler. A part of the compiler called the processor deals with these directives before it begins the real compilation process.The preprocessor directive #include tells the compiler to insert another file into your source file. In effect, the #include directive is replaced by the contents of the file indicated. Using an #include directive to insert another file into your source file is similar to pasting a block of text into a document with your word processor. #include is only one of many preprocessor directives,all of which can be identified by the initial # sign. The use of preprocessor directives is not as common in C++ as it is in C, but we'lllook at a few additional examples as we go along. The type file usually included by #include is called a header file.

HEADER FILE:

The preprocessor directive #include tells the compiler to add the source

file **IOSTREAM** to the program's source file before compiling.

- The newer Standard C++ header files don't have a file extension, but some older header files, left over from the days of the C language, havethe extension .h .