OOPS in C++

Why Object-Oriented Programming?

- C++ language was designed with the main intention of adding object-oriented programming to C language
- As the size of the program increases readability, maintainability, and bug-free nature of the program decrease.
- This was the major problem with languages like C which relied upon functions or procedure (hence the name procedural programming language)
- As a result, the possibility of not addressing the problem adequately was high
- Also, data was almost neglected, data security was easily compromised
- Using classes solves this problem by modeling program as a real-world scenario

Difference between Procedure Oriented Programming and Object-Oriented Programming

Procedure Oriented Programming

→ Consists of writing a set of instruction for the computer to follow

→ The main focus is on functions and not on the flow of data

→ Functions can either use local or global data

→ Data moves openly from function to function

Object-Oriented Programming

→ Works on the concept of classes and object

→ A class is a template to create objects

→ Treats data as a critical element

→ Decomposes the problem in objects and builds data and functions around the objects

Classes in C++

✓ Classes are user-defined data-types and are a template for creating objects. Classes consist of variables and functions which are also called class members.

Basic Concepts in Object-Oriented Programming

- **Classes -** Basic template for creating objects
- **Objects –** Basic run-time entities
- **Data Abstraction & Encapsulation –** Wrapping data and functions into a single unit
- **Inheritance –** Properties of one class can be inherited into others
- **Polymorphism –** Ability to take more than one forms
- **Dynamic Binding –** Code which will execute is not known until the program runs
- **Message Passing –** message (Information) call format

Benefits of Object-Oriented Programming

- ✓ Better code reusability using objects and inheritance
- ✓ Principle of data hiding helps build secure systems
- ✓ Multiple Objects can co-exist without any interference
- ✓ Software complexity can be easily managed

Public Access Modifier in C++

All the variables and functions declared under public access modifier will be available for everyone. They can be accessed both inside and outside the class. Dot (.) operator is used in the program to access public data members directly.

Private Access Modifier in C++

All the variables and functions declared under a private access modifier can only be used inside the class. They are not permissible to be changed by any object or function outside the class.

```
class Employee{
    private:                 }
       int a, b, c;          }
       void func1(){         }
          ……..}              }  class definition
    public:                  }
        int d, e;            }
       void func2(void);     }      function of a class can also be
 };                                 declared outside the class like this
void Employee::func(void) ──────────↗
Employee e1; ────→ object e1 is created.
```

Why use classes instead of structures?

→The default mode in structures is public whereas default mode is private in classes.

Note:-

- You can declare objects along with the class declarion like this:

```
class Employee{
 ….. Class definition…….
} harry, rohan, lovish;
```

- harry.salary = 8 makes no sense if salary is private
- We can also define an array as a class member.

Nesting of Member Functions

If one member function is called inside the other member function of the same class it is called nesting of a member function.

Objects Memory Allocation in C++

- The memory is only allocated to the variables of the class when the object is created. The memory is not allocated to the variables when the class is declared.
- Single variables can have different values for different objects, so every object has an individual copy of all the variables of the class.
- But the memory is allocated to the function only once when the class is declared. So the objects don't have individual copies of functions only one copy is shared among each object.

Static Data Members in C++

→When a static data member is created, there is only a single copy of the data member which is shared between all the objects of the class. If the data members are not static then every object has an individual copy of the data member and it is not shared.

Static Methods in C++

→ When a static method is created, they become independent of any object and class.
→ Static methods can only access static data members and static methods.
→ Static methods can only be accessed using the scope resolution operator.

e.g.

```
class Employee
{
    static int count;   //default value is zero
public:
    void setData(void)
    { count++;}

    static void getCount(void){
        // cout<<id; → throws an error
        cout<<count<<endl;
    }
};
int main()
{
    Employee harry, rohan, lovish;
    Employee::getCount(); →static method is called using ::
                           operator
}
```

→ Every time setData function is invoked by the different objects, the value of count will resume from previous value, there will be no separate count for different objects

Array of Objects in C++

An array of objects is declared in the same way as any other data-type array. An array of objects consists of class objects as its elements.

```
class Employee
{
  ……………..
};
Employee fb[4];
```

Passing Object as Function Argument

Objects can be passed as function arguments. This is useful when we want to assign the values of a passed object to the current object.

```
 class complex{
    int a;
    int b;
   public:
     void setDataBySum(complex o1, complex o2){
            a = o1.a + o2.a;
            b = o1.b + o2.b; }
  };

int main(){
    complex c1, c2, c3;
    c3.setDataBySum(c1, c2);
}
```

Friend Functions →

- Friend functions are those functions that have the right to access the private data members of class even though they are not defined inside the class.
- It is necessary to write the prototype of the friend function inside the class as :

```
friend Complex sumComplex(Complex o1,Complex o2);
```

→ Above statement tells the compiler that function sumComplex is my friend and can access my members.

Properties of friend function :-

1. The friend function will not become a member of the class. Hence it can’t be invoked by objects of the class. But a friend function can return an object of

that that class type as in above case friend function sumComplex returns of the type Complex.

2. A friend function can be invoked without the help of any object and it usually contain objects as arguments
3. Can be declared under the public or private access modifier, it will not make any difference.
4. It cannot access the members directly by their names, it needs (object_name.member_name) to access any member.
5. A function can be declared as friend function in two separate classes – but a forward declaration of second class should be given if necessary.

Member Friend Function:-

→A class can declare a function as friend which is a member of other class, but only that function can access the private members not the other functions.

syntax :

```
friend int Calculator::sumRealComplex(Complex, Complex);
```

→where sumRealComplex is a function of class Calculator.

→A class can declare multiple functions as friend.

Friend Classes:-

→Friend classes are those classes in which all of its functions have the permission to access private members of the class in which they are declared.

Syntax → *friend class Calculator;*

✓ If class calculator is built before the class in which it is declared as friend, then we need to write a forward declaration of that class before the class calculator.

NOTE:- We can also swap the members of two different classes by a swapping function by passing the address of objects.

Constructor →

→ Constructor is a special member function with the same name as of the class.
→ It is used to initialize the objects of its class.
→ It is automatically invoked whenever an object is created.
→ It should be declared in the public section of the class.
→ They cannot return values and do not have return types.
→ It can have default arguments.
→ We cannot refer to their address.

```
Complex(void);   // Constructor declaration inside a class

Complex ::Complex(void) //--->
```

This is a default constructor as it takes no parameter

```
{   a = 0;
    b = 0;
}
```

$\Big\}$ outside class

```
Complex ::Complex(int x, int y)-->
    {a = x; b = y;}
```

This is a parameterized constructor as it takes 2 arguments.

Now, in the main function constructor will be called when a object is created in any of two ways :

i. Implicit call

```
Complex a(4, 6);
```

ii. Explicit call

```
Complex b = Complex(5, 7);
```

Some Important Concepts

- **Constructor Overloading:-** Constructors can also be overloaded by having different numbers of parameters or different types of parameter.

- A constructor can have **default parameter.**

- **Dynamic initialization:-** Objects can be initialized at the runtime of program by depending on the input from user. This can be achieved with the help of overloading.

Note:- We need to make sure that we create a default constructor,

I. Before creating the overloaded constructors for dynamic initialization so that compiler may not get confused. We can avoid this practice by instantiating object after taking input.
(see progam for better understanding)

II. If an object is instantiated with no argument and the class contains a parameterized constructor because whenever an object is created the constructor gets invoked for sure.

→ A class, by default, always has its own constructor which is invoked at the moment whenever an object is created. But if we create a constructor manually then the class doesn't create its own constructor. That's why we create a default constructor in the case of creating an object with no argument so as to avoid error.

→An object is initialized like:

Complex obj = 3; → 3 will be passed as an argument to constructor

✓ Constructor is invoked when an object is created, not when the object is initialized to any value after the creation.

```
class Complex
{
    int a, b;

public:
    Complex()
       {a = 0; b =0;}
    Complex(int x, int y)
      {a = x; b = y;}
    Complex(int x)
      {a = x; b = 0;}

    void printNumber(){
   cout << "Your number is " << a << " + " << b << "i" << endl;
    }
};
```

```
int main(){
    Complex c1(4, 6);
    c1.printNumber();

    Complex c2(5);
    c2.printNumber();

    Complex c3;
    c3.printNumber();
    return 0;
}
```

- **Copy Constructor**:- A constructor used to create an object which is a copy of another object by passing its reference to constructor.

➢ It is created as:

```
Number(Number &obj){
cout<<"Copy constructor called!!!"<<endl;
a = obj.a;}
```

➢ It can be invoked by : `Number z1(z);`
➢ If & is not used in it, recursion keep occurring.
➢ It can also be invoked by assignment operator but object should only be created while assignment of the object to new object
➢ When no copy constructor is found, compiler supplies its own copy constructor.
➢ Hence compiler creates two types of constructors: default constructor and copy constructor --> when they are not created by the programmer.

Destructor →

- A destructor is used to free up the space used by object.
- It is invoked for an object when the compiler get to know that the object is no longer is needed in the program.
- Destructor never takes an argument nor does it return any value.
- It can’t be static.
- For a class num it is declared inside public as:

```
~num(){
  … … …
  }
```

- It doesn’t destroy the object, however it is the last object in the life span of an object.