CHAPTER # 8

Classes and Objects

Object Oriented Programming (OOP)

- It refers to a programming method that is based on objects.
- An object can be considered a thing that can perform a set of activities and have different properties i.e. student, vehicle, building etc.
- OOP provides a set of rules for defining and managing objects.
- In C++, classes and objects are the important features of object oriented programming.

Class

- A class is a user-defined data type that is used to create objects.

OR

- A **class** is a user defined Blueprint or Model from which **objects** are created.
- For example, a class **Human** can be used to define the characteristics and functions of a human.

Member of a Class

- A class consists of the following members:
    1. Data Member
    2. Member Functions
- The variables defined inside a class are called **data members**. It is also known as attribute.
- The functions defined inside a class are called **member functions**. The functions are used to perform different operations on the attributes.

Declaring a Class

- The keyword **class** is used to declare a class. The syntax of declaring a class is as follows:

```
class class_name
{
        body
};
```

- For example:

```
class test
{
        int a;
        char c;
        float x;
        void show()
        {
                cout<<a;
                cout<<c:
                cout<<x;
        }
        void input()
        {
                cin>>x;
                cin>>a;
                cin>>c;
        }
};
```

Access Specifier

- It is used to specify the access level of class members. Different access specifiers in C++ are as follows:

1. **The private Access Specifier:**
   - The private access specifier tells the compiler that class member can only be accessed within a class. It cannot be accessed from outside the class.
   - The data member are normally declared with private access specifier because the data of an object is more sensitive.
   - The private access specifier protects the data member from direct access from outside the class.
2. **The public Access Specifier:**
   - The public access specifier tells the compiler that the class member can be accessed from anywhere in the programm.

Example:

```
class test
{
        private:
                int a;
                char c;
                float x;
        public:
                void show()
                {
                        cout<<a;
                        cout<<c:
                        cout<<x;
                }
                void input()
                {
                        cin>>x;
                        cin>>a;
                        cin>>c;
                }
};
```

Object

- Objects are instances of class, which holds the data members declared in class and the member functions.

OR

- A variable of type class is called object.
- The process of declaring a variable of type class is called instantiating the class and the variable itself is called instance or object of the class.
- A class cannot be used without creating its object.
- The general syntax for creating an object:

  class_name Object_name;

- The definition of a class does not occupy any memory, it only defines what the class looks like.
- When an object is created, then memory is set aside for all the data members and member functions of that class.
- To access members of an object, whether data members or member functions, dot operator (**.**) is used with the member name.

  Object_name . Member_name;

Example # 1:

```
#include <iostream.h>
#inlucde <conio.h>
class rectangle
{
	private:
		int length;
		int width;
	public:
		void input()
		{
			cout<<“Enter Length: “;
			cin>>length;
			cout<<“Enter Width: “;
			cin>>width;
		}
		void area()
		{
			cout<<“The Area of Rectangle is “<<length*width;
		}
};
void main()
{
	clrscr();
	rectangle R1;
	R1.input();
	R1.area();
	getch();
}
```

Example # 2

- Write a program using a class to input two values using a member functions of a class. Display the sum of two values by using another member function of the class.

```
#include <iostream.h>
#inlucde <conio.h>
class add
{
	private:
		int a;
		int b;
	public:
		void input()
		{
			cout<<“Enter First number: “;
			cin>>a;
			cout<<“Enter Second number“;
			cin>>b;
		}
		void sum()
		{
			cout<<“Sum of two Numbers is: “<< a + b;
		}
};
void main()
{
	clrscr();
	add obj;
	obj.input();
	obj.sum();
	getch();
}
```

Program # 1:

- Write a C++ program implementing a class with the name Circle having two functions with the names: GetRadius and CalArea. The functions should get the value for the radius from the user and then calculate the area of the circle and display the results.

```
#include <iostream.h>
#inlucde <conio.h>
class Circle
{
	private:
		float  radius;
	public:
		void Get_radius()
		{
			cout<<“Enter Radius“;
			cin>>lradius;
		}
		void Cal_area()
		{
			cout<<“The Area of Circle is “<<3.14 * radius * radius;
		}
};
void main()
{
	clrscr();
	Circle c;
	c.Get_radius();
	c.Cal_area();
	getch();
}
```

Data Hiding/Encapsulation

- The process of protecting the data members and member functions of a class against illegal access from outside the class is called data hiding or data encapsulation.
- In c++, data hiding or encapsulating is achieved through different access level i.e. private, public and protected.
- Private data members and member functions can only be accessed by the members of the same class and cannot be accessed from outside the class.

Constructor

- Constructor is a special type of member function that initializes an object automatically when it is created.
- It is called automatically when an object is created.
- It is typically used to initialize the data members in the class.
- Some rules for the constructor are as follows:
  - The constructor has the same name as that of the class.
  - Constructor has no return data type (not even void).
  - Constructor is always public.

Example:

```
class test
{
        private:
                int a;
                float x;
        public:
                test()
                {
                        cout<<"I am constructor");
                        a=0;
                        x=0;
                }
                void show();
                {
                        cout<<a;
                        cout<<x;
                }
                void input();
                {
                        cin>>x;
                        cin>>a;
                }
};
```

Default Constructor / Implicit Constructor

- The default constructor that be called with no arguments or with default values given for all arguments.
- It is also called implicit constructor.
- The compiler declares a default constructor with no arguments if the user does not define any constructor for the class.

User-define constructor

- It is a constructor that is defined by the user for their own purposes.
- The compiler does not define any default constructor if user define constructor is exist in the class.
- It is especially used for initialization of data members.
- It is also called explicit constructor.

Example:

```
#include <iostream.h>
#inlucde <conio.h>
class rectangle
{
        private:
                int length;
                int width;
        public:
                rectangle(int a, int b)
                {
                        length = a;
                        width = b;
                }
                void input()
                {
                        cout<<“Enter Length: “;
                        cin>>length;
                        cout<<“Enter Width: “;
                        cin>>width;
                }
                void area()
                {
                        cout<<“The Area of Rectangle is “<<length*width;
                }
};
void main()
{
        clrscr();
        rectangle R1(0, 0);
        rectangle R2 (5, 4);
        R2.area();
        R1.area();
        getch();
}
```

Constructor Overloading

- The process of declaring multiple constructor with the same name but different parameters is known as constructor overloading.
- The constructor with same name must differ in one of the following ways:
  - Number of parameters.
  - Type of parameters.
  - Sequence of parameters.
- When an object is created, the compiler executes a constructor that matches the argument in function call.

Example:

```
#include <iostream.h>
#inlucde <conio.h>
class rectangle
{
        private:
                int length;
                int width;
        public:
                rectangle()
                {
                                length = 0;
                                width =0;
                }
                rectangle(int a, int b)
                {
                                length = a;
                                width = b;
                }
                void input()
                {
                                cout<<“Enter Length: “;
                                cin>>length;
                                cout<<“Enter Width: “;
                                cin>>width;
                }
                void area()
                {
                                cout<<“The Area of Rectangle is “<<length*width;
                }
};
```

```
void main()
{
   clrscr();
   rectangle R1;
   rectangle R2 (5, 4);
   R2.area();
   R1.area();
   getch();
}
```

Destructor

- A type of member function that is automatically executed when an object is destroyed is called destructor.
- Destructor fulfills the opposite functionality of constructor.
- It de-allocates memory that is allocated to an object during its creation.
- Destructor has the following specific features:
  - It has the same name as that of the class, precede by tild symbol (~).
  - It cannot take arguments.
  - It has no return type.

Example

```
#include <iostream.h>
#inlucde <conio.h>
class rectangle
{
        private:
                int length;
                int width;
        public:
                rectangle(int a, int b)
                {
                        length = a;
                        width = b;
                }
                ~rectangle()
                {
                        cout<<"Destructor is called "<<endl;
                }
                void input()
                {
                        cout<<"Enter Length: ";
                        cin>>length;
                        cout<<"Enter Width: ";
                        cin>>width;
                }
                void area()
                {
                        cout<<"The Area of Rectangle is "<<length*width;
                }
};
void main()
{
        clrscr();
        rectangle R1(0, 0);
        rectangle R2 (5, 4);
        R2.area();
        R1.area();
        getch();
}
```

Program # 3

- Write a C++ program implementing a class with the name ConstDest having a constructor and destructor functions in its body.

```
#include <iostream.h>
#inlucde <conio.h>
class ConstDest
{
    public:
        ConsDest()
        {
            cout<<“Constructor is executed”<<endl;
        }
        ~ConsDest()
        {
            cout<<“Destructor is executed”<<endl;
        }
    };
void main()
{
    clrscr();
     ConstDest a, b;
    getch();
}
```

Program # 4:

- Write a C++ program implementing a class with the name Time. This class should have a constructor to initialize the time, hours, minutes and seconds, initially to zero. The class should have another function name ToSecond to convert and display the time pass by the object into seconds.

```
#include <iostream.h>
#inlucde <conio.h>
class Time
{
        private:
                int h, m, s;
        public:
                Time
                {
                        h = m = s = 0;
                }
                void setTime( int hh, int mm, int ss)
                {
                        h = hh;
                        m = mm;
                         s = ss;
                }
                void ToSecond()
                {
                        long sec;
                        sec = s + (m * 60) + ( h * 3600 );
                        cout<< " Total Seconds = "<<sec<<endl;
                }
};
void main()
{
        clrscr();
        Time x;
        x.setTime(1, 5, 45);
        x.ToSecond();
        getch();
}
```

Inheritance

- A programming technique in which new classes are created from existing classes is called inheritance.
- The new class inherits all the properties and functions of original class.
- Inheritance uses the concept of parent and child class.
- The class from which other classes are created are known as **parent class** or **base class**.
- The class that inherit all the characteristics of other class are called **subclass** or **derived class** or **child class**.
- Example:
  - A class Vehicle can be represented into Bus, Truck, car etc. which have some common features like wheels, registration no., model etc. The subclasses have their own specific features like Bus has the seats for the people, truck has the space to carry good.
  - A class patient can be represented into Indoor and Outdoor patient. Both have some common features like name, father name, age, address, diseases etc. but indoor patient have Ward no. and Bed no. as its unique features and the outdoor patient have “next date to visit” as its unique feature.
- **Protected Access Specifier:**
  - It provide the same level of access as t hat of private but protected members of a class can also be accessed in child classes which are called derived classes.

Syntax of using inheritance in classes

```
class A
{
        members of class A;
};
class B : public A
{
        members of Class B;
};
```

- Class B is publically derived from class A. Class B has its own members and inherits all the members of Class A. Class B has the rights to use all the members of Class A.

Program # 2:

- Write a C++ program implementing inheritance between Employee (base class) and Manager (Derived Class).

```
#include <iostream.h>
#inlucde <conio.h>
class Employee
{
        protected:
                int EmpID;
                int Salary;
        public:
                void setEmp()
                {
                        cout<<“Enter Employee ID: “;
                        cin>>EmpID;
                        cout<<“Enter Salary: “;
                        cin>>Salary;
                }
                void showEmp()
                {
                        cou<<“Employee ID: “<<EmpID<<endl;
                        cout<<“Salary: “<<Salary<<endl;
                }
};

class manager : public Employee
{
        private:
                int DeptID;
        public:
                void setM()
                {
                        cout<<“Enter Department ID: “;
                        cin>>DepID;
                }
                void showM()
                {
                        cout<<“Department ID: “<<DeptID<<endl;
                }
};
```

```
void main()
{
   clrscr();
   Manager M;
   M.setEmp();
   M.setM();
   M.showEmp();
   M.showM();
   getch();
}
```

Polymorphism

- It is the ability to use an operator or function in multiple ways.
- The word **poly** means **many** and **morphism** means **form**.
- In C++, polymorphism can be achieved by one of the following concepts:
  - Function overloading
  - Operator overloading
  - Virtual functions

- **Function Overloading:**
  - The process of declaring multiple functions with same name but different parameters is called function overloading.
- **Operator Overloading:**
  - The process of defining additional meanings of the operator is called operator overloading.
  - It enables the operator to perform different operations depending on the type of operands.
  - For example, the addition operator (+) is used to add numeric values. The operator overloading can enable it to manipulate two strings.
- **Virtual Functions:**
  - A function that is defined in the parent class and is re-defined (overriden) by derived class is called virtual function.