FAST NATIONAL UNIVERSTIY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

DEPARTMENT OF COMPUTER SCIENCE

CL217 – OBJECT ORIENTED PROGRAMMING LAB

<u>CLASSES AND OBJECTS IN C++</u>

LAB MANUAL # 05

Instructor: Mazhar Iqbal

SEMESTER SPRING 2023

OBJECT ORIENTED PROGRAMMING LANGUAGE

Table of Contents

Object Oriented Programming (OOP)

OOP is methodology or paradigm to design a program using class and object.

OOP is paradigm that provides many concepts such as:

- Classes and objects
- Encapsulation (binding code and its data) etc.
- Inheritance
- Polymorphism
- Abstraction
- Overloading

Class

- ❖ A Class is a collection of data and functions. The data items and functions are defined within the class. Functions are written to work upon the data items and each function has a unique relationship with data items of the class.
- ❖ Classes are defined to create user defined data types. These are similar to built-in data types available in all programming languages.
- ❖ Definition of data type does not create any space in the computer memory. When a variable of that data type is declared, a memory space is reserved for that variable.
- ❖ Similarly, when a class is defined, it does not occupy any space in the computer memory. It only defines the data items and the member function that can be used to work upon its data items. Thus, defining a class only specifies its data members and the relationship between the data items through it functions.

Defining a class

A class is defined in a similar way as structure is defined. The keyword “**class**” is used to define the class. The general syntax to define a class is:

```
class class_name

{

body of the class;

} ;
```

class is a keyword that is used to define a class.

class_name It represents the name of the class.

body of classs The body of the class consist of the data items and the functions. These are called members of the class. These are written between braces.

Semicolon (;) The body of a class ends with semicolon.

Members of a class

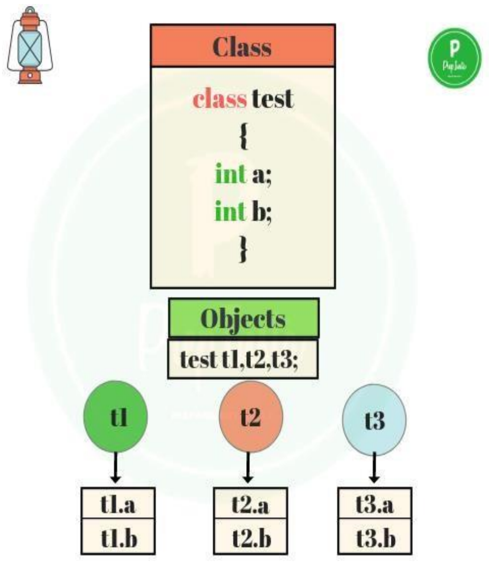
A class contains data items and functions. These are called members of the class. The data items are called data members and the functions are called member functions.

1. Data Members

The data items of a class are called data members of the class. For example, a class that has four integer type and two float type data items is declared as:

```
class abc

{

        int w , x , y , z;

        float a , b;

}
```

In the above class **a, b, w, x, y** and **z** are data members of the class “**abc**”.



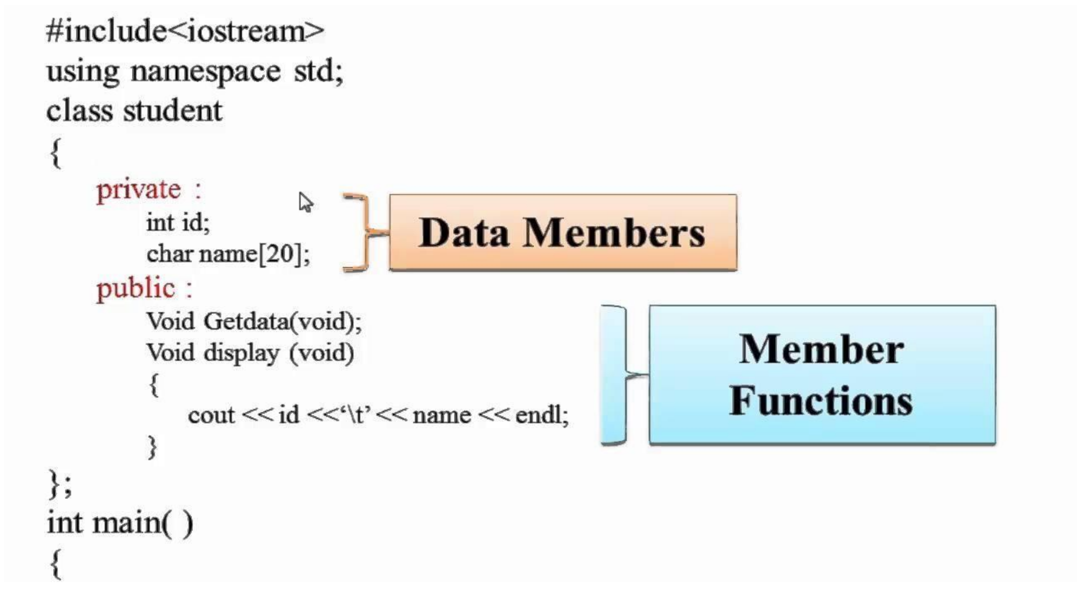
2. Member Functions

The functions of a class that are defined to work on its data members are called member functions of the class. The member functions may be defined within the class or outside it.

For example:

```
class xyz
{
	private:
	int a , b , c;
	public:
	void getData(void)
	{
		cout<<Enter value of a, b and c”;
		cin>>a>>b>>c;
	}
	void printData(void)
	{
		cout<<“a= ”<<a<<endl;
		cout<<“b= ”<<b<<endl;
		cout<<“c= ”<<c<<endl;
	}
} ;
```

In this class, there are three data members and two member functions. The member functions are “getData” and “printData”. The “getData” function is used to input values into data members a, b and c. The “printData” function is used to print values of the data members on the computer screen.



```
#include<iostream>
using namespace std;
class student
{
    private :
        int id;
        char name[20];
    public :
        Void Getdata(void);
        Void display (void)
        {
            cout << id <<‘\t’ << name << endl;
        }
};
int main( )
{
```

Objects

- A data type is used to declare a variable. A variable of a data type is also known as the instance or case of that data type.
- Each variable has unique name but each variable follows the rules of its data type. When a variable of a data type is declared, some space is reserved for it in the memory.
- A class is also like a data type. It is therefore used to declare variables or instances. The variables or instances of a class are called **objects**.
- A class may contain several data items and functions. Thus, the object of a class consists of both the **data members** and **member functions** of the class.
- The combining of both the data and the functions into one unit is called data **encapsulation**.
- An object represents data members of a class in the memory. Each object of class has unique name. The name of an object differentiates it from other objects of the same class.
- The values of data members of different objects may be different or same. The values of data members in an object are known as the **state** of the object.
- The functions in an object are called the member functions. They are also known as the them **methods**.
- The member functions are used to process and access data of the objects.

Declaring object of a class

- The objects of a class are declared in the similar way as the variables of any data or structure type are declared.
- When a class is defined, no space is reserved for it in the memory. It only provides information how its object will look like.
- When an object of a class is declared, a memory space is reserved for that object.
- The syntax to create objects of a class type is:
    - **class_name object_name separated by commas;**
- For example, to define an objects of Student class, the statement is written as:
    - **Student s1;**
- In the above statement one object namely **s1** is declared of Students class. It is the declaration of an object that actually creates an object in the memory.

A Class is a Full-Fledged Type

- ❖ A class is a type just like int and double. You can have variables of a class type, you can have parameter of class type, a function can return a value of a class type, and more generally, you can use a class type like any other type.

  For example: Student s1,s2,s3;

Accessing data members and member functions

- ❖ The data members and member functions of class can be accessed using the dot('.') operator with the object.
- ❖ For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

Accessing Data Members

- ❖ The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object.
- ❖ Accessing a data member depends solely on the access control of that data member. This access control is given by Access modifiers in C++.
- ❖ There are three access modifiers: public, private and protected.

Accessing Member Functions

- ❖ The member functions of a class is called or accessed in similar way as member or data item of a structure is called.
- ❖ The member function is called/accessed through an object of the class.
- ❖ The dot operator is used. The dot operator connects the **object name** and **member function**.
- ❖ For example, if "**add**" is the name of the object and "**pdate()**" is the member function then the member function is called as shown below:

  add.pdate();
- ❖ The **dot operator** is also called the class member **access operator**.
- ❖ Only those member functions can be accessed from outside the class with dot operator that have been declared as public.

Example

```
// C++ program to demonstrate accessing of data members
#include <iostream>
using namespace std;

#include<iostream>
using namespace std;
class Rectangle
{
   private:
   int length;
   int width;
     public:
   Rectangle()
   {
          length=0;
          width=0;
   }

   Rectangle(int l, int w)
   {
          length=l;
          width=w;
   }

   int getLength()
   {
          return length;
   }

   int getWidth()
   {
          return width;
   }

   void setLength(int l)
   {
          length=l;
   }

   void setWidth(int w)
   {
          width=w;
   }

     int calc_Area()
   {
           int area=length*width;
           return area;
   }
};
int main () {
     int length, width;
     cout<<"Enter length: "<<endl;
     cin>>length;
     cout<<"Enter width: "<<endl;
     cin>>width;
     Rectangle r;
     r.setLength(length);
     r.setWidth(width);
     cout<<"\nLength: "<<r.getLength();
     cout<<"\nWidth: "<<r.getWidth();
     cout<<"\nArea: "<<r.calc_Area();
   return 0;
}


/*
Output
Enter length:
2
Enter width:
3

Length: 2
Width: 3
Area: 6*/
```

Default/ Non parameterized and Parameterized constructor

```
#include<iostream>
using namespace std;
class Rectangle
{
   private:
   int length;
   int width;
   public:
     Rectangle() // default constructor
   {
           length=0;
           width=0;
   }
   Rectangle(int l, int w) //parametarized constructor
   {
           length=l;
           width=w;
   }

   int getLength()
   {
           return length;
   }

   int getWidth()
   {
           return width;
   }

   void setLength(int l)
   {
           length=l;
   }

   void setWidth(int w)
   {
           width=w;
   }

     int calc_Area()
   {
           int area=length*width;
           return area;
   }
};
int main () {
     int length, width;
     cout<<"Enter length: "<<endl;
     cin>>length;
     cout<<"Enter width: "<<endl;
     cin>>width;
     Rectangle r1(length, width); // calling the parameterized
constructor
     cout<<"\n\nLength of R1: "<<r1.getLength();
     cout<<"\nWidth of R2: "<<r1.getWidth();
     cout<<"\nArea of R1: "<<r1.calc_Area();
     Rectangle r2; // default constructor calls automatically
     cout<<"\n\nLength of R2: "<<r2.getLength();
```

```
      cout<<"\nWidth of R2: "<<r2.getWidth();

   return 0;
}

/*
Output
Enter length:
2
Enter width:
3


Length of R1: 2
Width of R2: 3
Area of R1: 6

Length of R2: 0
Width of R2: 0*/
```

Passing Object to a function (by Value)

```
#include<iostream>
using namespace std;
class Rectangle
{
  private:
  int length;
  int  width;
  public:
          Rectangle() // default constructor
  {
                  length=0;
                  width=0;
  }
  Rectangle(int l, int w) //parametarized constructor
  {
                  length=l;
                  width=w;
  }

  int getLength()
  {
                  return length;
  }

  int getWidth()
  {
                  return width;
  }

  void setLength(int l)
  {
                  length=l;
  }

  void setWidth(int w)
  {
                  width=w;
  }
};
int calc_Area(Rectangle r)
{
          int area=r.getLength()*r.getWidth();
          return area;
}
```

```
int main () {
        int length, width;
        cout<<"Enter length: "<<endl;
        cin>>length;
        cout<<"Enter width: "<<endl;
        cin>>width;
        Rectangle r1(length, width); // calling the parameterized constructor
        cout<<"\n\nLength of R1: "<<r1.getLength();
        cout<<"\nWidth of R2: "<<r1.getWidth();
        cout<<"\nArea of R1: "<<calc_Area(r1);
        return 0;
}
```

Output:

```
Enter length:
2
Enter width:
4
Length of R1: 2
Width of R1: 4
Area of R1: 8
```

Passing Object to a function (by Reference)

```
#include<iostream>
using namespace std;
class Rectangle
{
  private:
  int length;
  int  width;
  public:
        Rectangle() // default constructor
  {
                length=0;
                width=0;
  }
  Rectangle(int l, int w) //parametarized constructor
  {
                length=l;
                width=w;
  }

  int getLength()
  {
                return length;
  }

  int getWidth()
  {
                return width;

  }

  void setLength(int l)
  {
                length=l;
  }

  void setWidth(int w)
  {
                width=w;
  }
};
void changeLength(Rectangle *r)
  {
                r->setLength(5);
  }
```

```
int main () {
        Rectangle r;
        int length, width;
        cout<<"\n\nLength: "<<endl;
        cin>>length;
        r.setLength(length);
        cout<<"\n\nWidth: "<<endl;
        cin>>width;
        r.setWidth(width);
        changeLength(&r);
        cout<<"\n\nNew Length: "<<r.getLength();
  return 0;
}
```

Output:

```

Length:
2


Width:
3


New Length: 5
```

Returning an Object from a Function

```
#include<iostream>
using namespace std;
class Rectangle
{
  private:
  int length;
  int  width;
  public:
          Rectangle() // default constructor
  {
                  length=0;
                  width=0;
  }
  Rectangle(int l, int w) //parametarized constructor
  {
                  length=l;
                  width=w;
  }

  int getLength()
  {
                  return length;
  }

  int getWidth()
  {
                  return width;

  }

  void setLength(int l)

  {

                  length=l;

  }

  void setWidth(int w)

  {

                  width=w;

  }

};
```

```
Rectangle create_Rect()
{
        int length, width;
        cout<<"Enter length: "<<endl;
        cin>>length;
        cout<<"Enter width: "<<endl;
        cin>>width;
        Rectangle r(length, width); // calling the parameterized constructor
        return r;
}
int main () {
        Rectangle r=create_Rect();
        cout<<"\n\nLength: "<<r.getLength();
        cout<<"\nWidth: "<<r.getWidth();
  return 0;
}
```

Output:

```
Enter length:
2
Enter width:
3


Length: 2
Width: 3
```