

OBJECT ORIENTED PROGRAMMING LAB



Lab Manual # 08

Classes and Objects in C++

Instructor: Muhammad Abdullah Orakzai

Semester Fall, 2021

Course Code: CL1004

**Fast National University of Computer and Emerging Sciences
Peshawar**

Department of Computer Science

OBJECT ORIENTED PROGRAMMING LANGUAGE

Table of Contents

Object Oriented Programming (OOP).....	1
Class	1
Defining a class.....	1
Members of a class	2
1. Data Members	2
2. Member Functions.....	2
Objects	4
Declaring object of a class.....	5
Accessing data members and member functions.....	6
Accessing Data Members.....	6
Example.....	6
Accessing Member Functions	7
Examples	7
Defining Member Functions outside class.....	13
Storage of Object in Memory.....	15
Functions vs Methods	17
Access Specifiers / Modifiers in C++	17
1. Public Access Specifier	18
Example 1: Public Access Specifier	18
2. Private Access Specifier	19
Example 1: Private Access Specifier	19
Example 2: Private Access Specifier	20
3. Protected Access Specifier	21
Example 1: Protected Access Specifier	21
Constructor in C++	23
1. Default constructor	23
2. Parameterized Constructor.....	26
The Default Copy Constructor	30
Example of The Default Copy Constructor.....	31

Constructor Overloading.....	33
Defining Constructor Outside Class	40
Defining Constructor Outside Class Example.....	40
References	41

Object Oriented Programming (OOP)

OOP is methodology or paradigm (نمونه) 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 its 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 class 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

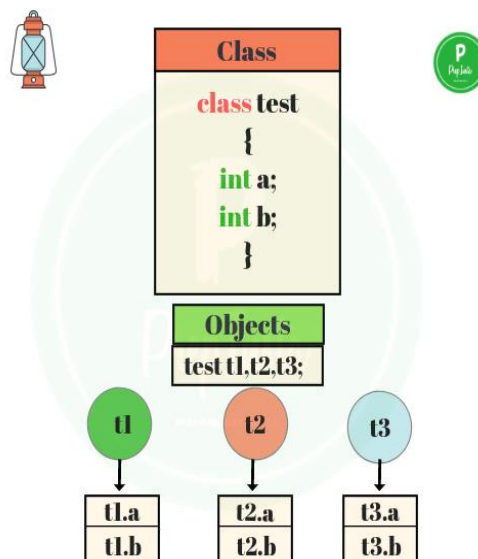
A class contains data items and functions. These are called members of the class. The data items are called data members and the functions are called member functions.

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

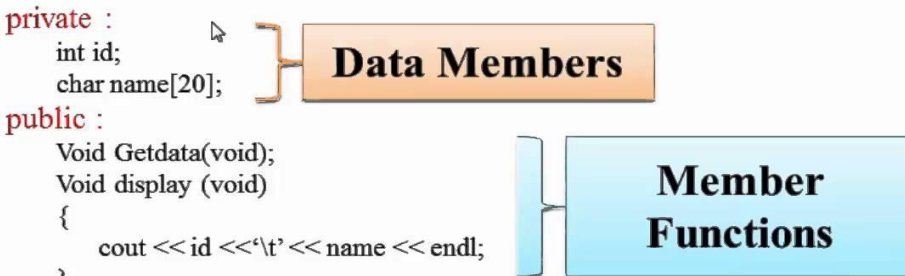
The functions of a class that are defined to work on its data members are called member functions of the class. The member functions may be defined within the class or outside it.

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.

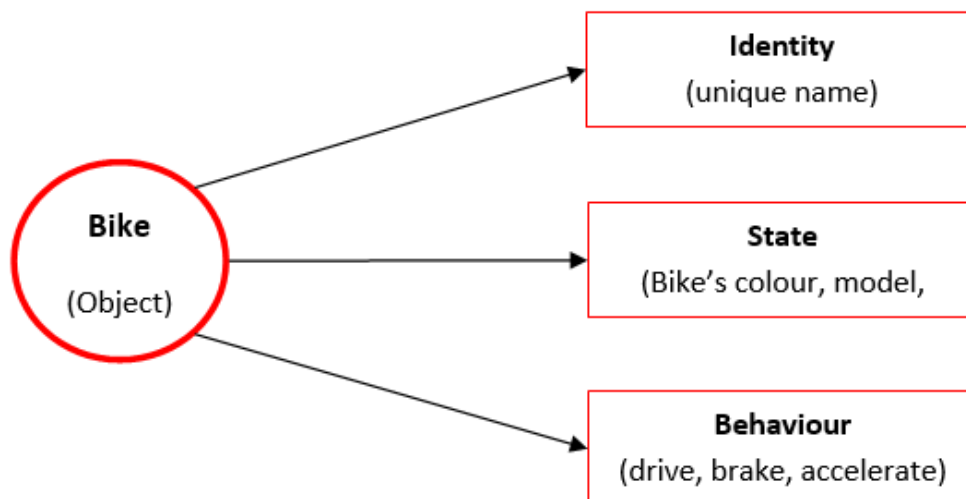
```
#include<iostream>
using namespace std;
class student
{
    private :
        int id;
        char name[20];
    public :
        Void Getdata(void);
        Void display (void)
        {
            cout << id << '\t' << name << endl;
        }
};
int main( )
{
```

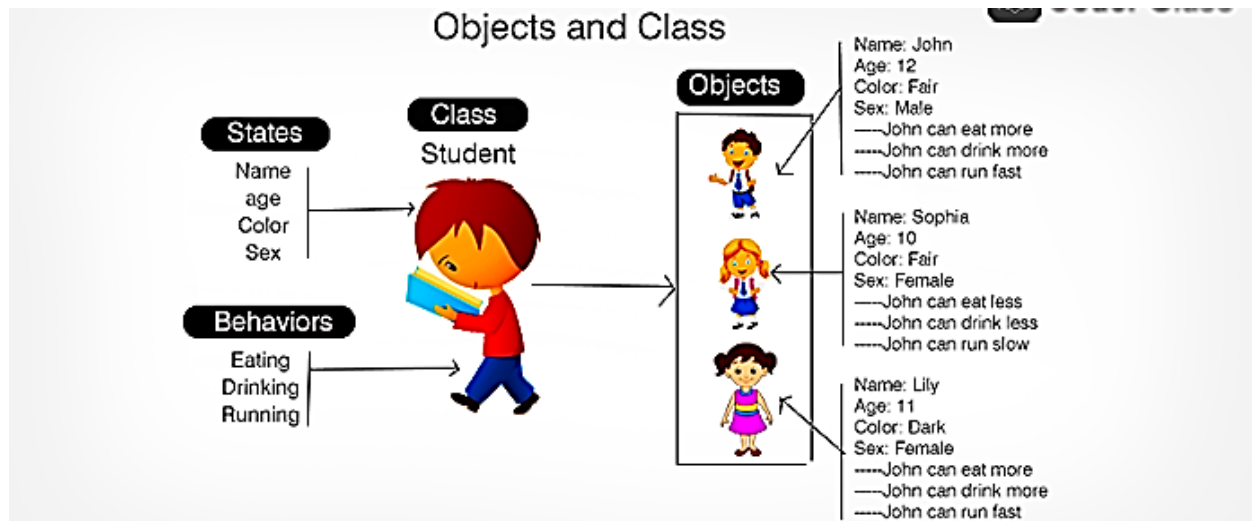


Objects

- ❖ 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 **methods**.
- ❖ The member functions are used to process and access data of the objects.

Characteristics of Object (Identity, State & Behavior)





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.

Example

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

class Student
{
    // Access specifier
    public:

    //Data members
    int id;
    string name;
    double salary;
}; // end of class body

int main () {

    // Declare an object of class Student
    Student s1;
    // accessing data members
    s1.id=144;
    s1.name="Aisha";
    s1.salary=60000;

    cout<<"Student Id is:"<<s1.id<<endl;
    cout<<"Student Name is:"<<s1.name<<endl;
    cout<<"Student Salary is:"<<s1.salary<<endl;
    return 0;
}
```

```
/*  
Output  
Student Id is:144  
Student Name is:Aisha  
Student Salary is:60000  
*/
```

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.

Examples

```
class Student  
{  
    // Access specifier  
    public:  
    //Data members  
    int id;  
    string name;  
    double salary;  
    // Member Functions  
    void setData(int i, string n, double s)  
    {  
        id=i;  
        name=n;  
        salary=s;  
    }  
  
    void printData()  
    {
```

```
        cout<<"Student Id is:"<<id<<endl;
        cout<<"Student Name is:"<<name<<endl;
        cout<<"Student Salary is:"<<salary<<endl;
    }
}; // end of class body
int main () {

    // Declare an object of class Student
    Student s1;

    // accessing member function
    s1.setData(144, "Aisha", 60000);

    s1.printData();

    return 0;
}

/*
Output
Student Id is:144
Student Name is:Aisha
Student Salary is:60000
*/
```

Program 01: Write a program to input a date and print on the screen using class.

```
#include <iostream>
using namespace std;

class Date
{
    // Access specifier
    private:
    //Data members
    int y,m,d;
    public:
    void getDate()
    {
        cout<<"Enter Year: "; cin>>y;
        cout<<"Enter Month: "; cin>>m;
        cout<<"Enter Day: "; cin>>d;
    }

    void printDate()
```

```

    {
        cout<<"Date is :";
        cout<<d<<"/" <<m<<"/"<<y;
    }
}; // end of class body

int main () {

    // Declare an object of class Date
    Date date;

    // accessing member function
    date.getDate();
    date.printDate();
    return 0;
}

```

```
/*
```

Output

```

Enter Year: 1995
Enter Month: 12
Enter Day: 12
Date is :12/12/1995
*/

```

Program 02: Write a program by using class to input values using a member functions of a class. Display the sum of two values by using another member function of the class.

```

#include <iostream>
using namespace std;

class Sum
{
    // Access specifier
    private:
    //Data members
    int n , m;

    public:
    void getDate(int x, int y)
    {
        n=x;
        m=y;
    }
}

```

```

    }

    void displayData()
    {
        cout<<"Sum is: "<<(n+m);
    }
}; // end of class body
int main () {

    // Declare an object of class
    Sum sum;

    int x, y;
    cout<<"Enter first No. :"; cin>>x;
    cout<<"Enter second No. :"; cin>>y;

    // accessing member function
    sum.getDate(x,y);
    sum.displayData();
    return 0;
}

```

```

/*

```

Output

```

Enter first No. :4
Enter second No. :4
Sum is: 8
*/

```

Program 03: Write a program to input the name of student and marks of three subjects, calculate the total marks and average marks. Each subject has maximum of 100 marks.

```

#include <iostream>
using namespace std;

class StudentRecord
{
    private:
        char name[15];
        float s1,s2,s3, total, avg;
    public:
        void getRecord()
        {
            cout<<"Enter Name of the student: "; cin>>name;

```

```

        cout<<"Enter marks of 1st subject: "; cin>>s1;
        cout<<"Enter marks of 2nd subject: "; cin>>s2;
        cout<<"Enter marks of 3rd subject: "; cin>>s3;
        total= s1+s2+s3;
        avg= total/3.0;
    }
    void displayRecord()
    {
        cout<<"Name of the student : "<<name<<endl;
        cout<<"Marks of 1st subject : "<<s1<<endl;
        cout<<"Marks of 2nd subject : "<<s2<<endl;
        cout<<"Marks of 3rd subject : "<<s3<<endl;
        cout<<"Total Marks : "<<total<<endl;
        cout<<"Average Marks : "<<avg<<endl;
    }
}; // end of class body

int main () {

    // Declare an object of class
    StudentRecord stdRecord;

    // accessing member function
    stdRecord.getRecord();
    stdRecord.displayRecord();
    return 0;
}

```

/*

Output

```

Enter Name of the student: Aousaf
Enter marks of 1st subject: 55
Enter marks of 2nd subject: 77
Enter marks of 3rd subject: 88
Name of the student : Aousaf
Marks of 1st subject : 55
Marks of 2nd subject : 77
Marks of 3rd subject : 88
Total Marks : 220
Average Marks : 73.3333
*/

```

Program 04: Write a program by using class Employee to input the record of employees.

Program 04: Write a program by using class Employee to input the record of employees.

Define the following data members:

- name, bpay, h_rent, ma, gpay

Define the following member functions:

- to input data in name and bpay
- to calculate h_rent, ma, gpay
- to print complete record on the computer Screen.

Where

h-rent = house rent= 60 %

ma = medical allowance = 20 %

Gpay = bpay + h_rent + ma

```
#include <iostream>
using namespace std;

class EmployeeRecord
{
    private:
        char name[15];
        float bpay, h_rent, ma, gpay;

    public:
        void getRecord()
        {
            cout<<"Enter Name of the employee: "; cin>>name;
            cout<<"Enter basic pay of employee: "; cin>>bpay;
        }
        void allow()
        {
            h_rent =bpay*60/100;
            ma= bpay*20/100;
            gpay=bpay+h_rent+ma;
        }
        void displayRecord()
```

```

    {
        cout<<"Name of the Employee : "<<name<<endl;
        cout<<"Basic Pay          : "<<bpay<<endl;
        cout<<"House Rent         : "<<h_rent<<endl;
        cout<<"Medical Allowance   : "<<ma<<endl;
        cout<<"Net Pay            : "<<gpay<<endl;
    }
}; // end of class body
int main () {

    // Declare an object of class
    EmployeeRecord empRecord;

    // accessing member function
    empRecord.getRecord();
    empRecord.allow();
    empRecord.displayRecord();
    return 0;
}

/*
Output
Enter Name of the employee: Aisha
Enter basic pay of employee: 30000
Name of the Employee      : Aisha
Basic Pay                  : 30000
House Rent                 : 18000
Medical Allowance         : 6000
Net Pay                    : 54000
*/

```

Defining Member Functions outside class

- ❖ Members functions of a class can also be defined outside the class. In this case, only the prototype of the member function is declared inside the class.
- ❖ The member functions are defined outside the class in the similar way as user defined functions are defined. However, the scope resolution operator (::) is used in the member function declarator to define the function of the class outside the class.
- ❖ The general syntax of member function definition outside the class is:

```

type class_name :: function_name (arguments)

{   body of function   }

```


- ❖ **Note:** The member function is defined outside its class if the body of the function definition is large. Otherwise the function definition should be defined inside the class.
- ❖ To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

Program 05: Write a program by using class Employee to input the record of employee by defining the function member outside the class.

```
#include <iostream>
using namespace std;
class EmployeeRecord
{
    private:
        char name[15];
        float bpay, h_rent, ma, gpay;

    public:
        void getRecord(void);
        void allow(void);
        void displayRecord(void);
}; // end of class

int main () {

    // Declare an object of class
    EmployeeRecord empRecord;

    // accessing member function
    empRecord.getRecord();
    empRecord.allow();
    empRecord.displayRecord();
    return 0;
}

// Definition of getRecord using scope resolution operator ::
void EmployeeRecord::getRecord()
{
    cout<<"Enter Name of the employee: "; cin>>name;
    cout<<"Enter basic pay of employee: "; cin>>bpay;
}

// Definition of allow using scope resolution operator ::
void EmployeeRecord::allow()
{
    h_rent =bpay*60/100;
    ma= bpay*20/100;
```

```

        gpay=bpay+h_rent+ma;
    }
// Definition of displayRecord using scope resolution operator ::
void EmployeeRecord::displayRecord()
{
    cout<<"Name of the Employee : "<<name<<endl;
    cout<<"Basic Pay          : "<<bpay<<endl;
    cout<<"House Rent          : "<<h_rent<<endl;
    cout<<"Medical Allowance    : "<<ma<<endl;
    cout<<"Net Pay              : "<<gpay<<endl;

}

/*
Output

Enter Name of the employee: Aisha
Enter basic pay of employee: 30000
Name of the Employee : Aisha
Basic Pay              : 30000
House Rent             : 18000
Medical Allowance      : 6000
Net Pay                : 54000

*/

```

Storage of Object in Memory

- ❖ When an object of a class is created, a space is reserved in the computer memory to hold its data members. Similarly, separate memory spaces are reserved for each class object.
- ❖ The member functions of a class are, however, stored at only one place in the computer memory.
- ❖ All objects of the class use the same member functions to process data.
- ❖ Therefore while, each object has separate memory space for data members, the member functions of a class are stored in only one place and are shared by all objects of the class.

Program 06:

```

#include <iostream>
using namespace std;
class Temp
{
    private:
        int x;
        float y;

```

```
public:
    void getData(void)
    {
        cout<<"Enter value of x : "; cin>>x;
        cout<<"Enter value of y : "; cin>>y;
    }
    void print(void)
    {
        cout<<"Entered value of x = "<<x<<endl;
        cout<<"Entered value of y = "<<y<<endl;
    }
}; // end of class body
int main () {

    // Declare an object of class
    Temp a, b;

    cout<<"Get data in object a"<<endl;
    a.getData();
    cout<<"Get data in object b"<<endl;
    b.getData();

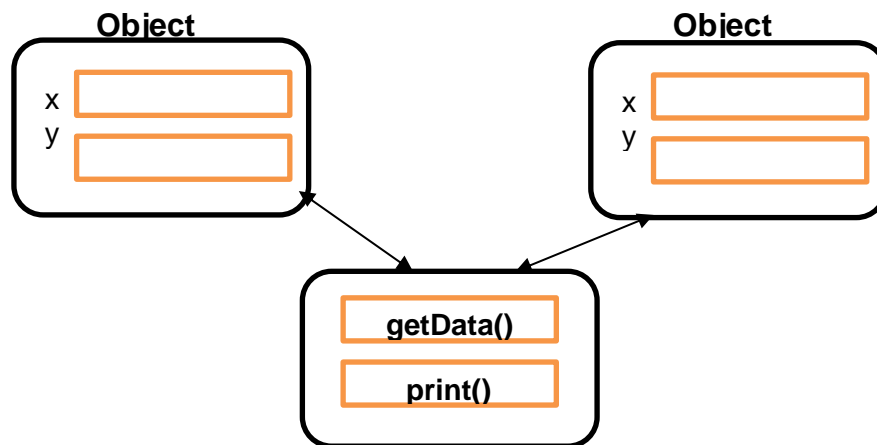
    cout<<"Data in object a is : "<<endl;
    a.print();
    cout<<"Data in object b is : "<<endl;
    b.print();

    return 0;
}
/*
```

Output

```
Get data in object a
Enter value of x : 44
Enter value of y : 5
Get data in object b
Enter value of x : 66
Enter value of y : 66
Data in object a is :
Entered value of x = 44
Entered value of y = 5
Data in object b is :
Entered value of x = 66
Entered value of y = 66
*/
```

- ❖ The storage of object **a** and **b** as mentioned in the above program example is shown below. These objects use the same member functions.



Functions vs Methods

Function — a set of instructions that perform a task.

Method — a set of instructions that are associated with an object.

METHODS

A method, like a function, is a set of instructions that perform a task. The difference is that a method is associated with an object, while a function is not.

Access Specifiers / Modifiers in C++

- ❖ It specifies that member of a class is accessible outside or not.
- ❖ Access modifiers are used to implement an important aspect of Object-Oriented Programming known as **Data Hiding**.
- ❖ Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

There are 3 types of access modifiers available in C++:

1. Public
2. Private
3. Protected

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be **Private**.

1. Public Access Specifier

- ❖ All the class members declared under the public specifier will be available to everyone.
- ❖ The data members and member functions declared as public can be accessed by other classes and functions too.
- ❖ The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

Example 1: Public Access Specifier

```
// C++ program to demonstrate public access modifier
#include<iostream>
using namespace std;

// class definition
class Circle
{
    public:
        double radius;

        double compute_area()
        {
            return 3.14*radius*radius;
        }
};

// main function
int main()
{
    Circle obj;

    // accessing public data member outside class
    obj.radius = 5.5;

    cout << "Radius is: " << obj.radius << "\n";
    cout << "Area is: " << obj.compute_area();
    return 0;
}

/*
Output
Radius is: 5.5
Area is: 94.985
*/
```

In the above program the data member *radius* is declared as public so it could be accessed outside the class and thus was allowed access from inside main().

2. Private Access Specifier

- ❖ The class members declared as *private* can be accessed only by the member functions inside the class.
- ❖ They are not allowed to be accessed directly by any object or function outside the class.
- ❖ Only the member functions or the friend functions are allowed to access the private data members of a class.
- ❖ friend functions will be discussed later

Example 1: Private Access Specifier

```
// C++ program to demonstrate private access modifier
#include<iostream>
using namespace std;
class Circle
{
    // private data member
    private:
        double radius;
    // public member function
    public:
        double compute_area()
        { // member function can access private
            // data member radius
            return 3.14*radius*radius;
        }
}; //end of class
// main function
int main()
{
    Circle obj; // creating object of the class
    // trying to access private data member
    // directly outside the class
    obj.radius = 1.5;
    cout << "Area is:" << obj.compute_area();
    return 0;
}
/*
Output
In function 'int main()': 11:16: error: 'double Circle::radius' is
private double radius; ^ 31:9: error: within this context obj.radius =
1.5; ^
*/
```

The output of above program is a compile time error because we are not allowed to access the private data members of a class directly outside the class. Yet an access to obj.radius is attempted, radius being a private data member we obtain a compilation error.

Example 2: Private Access Specifier

However, we can access the private data members of a class indirectly using the public member functions of the class.

```
// C++ program to demonstrate private access modifier
#include<iostream>
using namespace std;
class Circle
{
    // private data member
    private:
        double radius;
    // public member function
    public:
        void compute_area(double r)
        { // member function can access private
            // data member radius
            radius = r;
            double area = 3.14*radius*radius;
            cout << "Radius is: " << radius << endl;
            cout << "Area is: " << area;
        }
}; //end of class

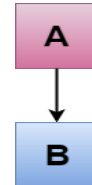
// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.compute_area(1.5);
    return 0;
}

/*
Output
Radius is: 1.5
Area is: 7.065
*/
```

3. Protected Access Specifier

- ❖ Protected access modifier is similar to private access modifier in the sense that it can't be accessed outside of its class unless with the help of friend class.
- ❖ The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.
- ❖ **Note:** This access through inheritance can alter the access modifier of the elements of base class in derived class depending on the **modes of Inheritance**.
- ❖ Where 'A' is the base class, and 'B' is the derived class.



Example 1: Protected Access Specifier

```
// C++ program to demonstrate protected access modifier
#include <iostream>
using namespace std;
// base class
class Parent
{
    // protected data members
    protected:
    int id_protected;
}; //parent class ends
// sub class or derived class from public base class
class Child : public Parent
{
    public:
    void setId(int id)
    {
        // Child class is able to access the inherited
        // protected data members of base class

        id_protected = id;
    }
    void displayId()
    {
        cout << "id_protected is: " << id_protected << endl;
    }
}; // child class ends
```



```
// main function
int main() {

    Child obj1;
    // member function of the derived class can
    // access the protected data members of the base class
    obj1.setId(81);
    obj1.displayId();
    return 0;
} // end of main() function

/*
Output
id_protected is: 81

*/
```

Summary: public, private, and protected

public elements can be accessed by all other classes and functions.

private elements cannot be accessed outside the class in which they are declared, except by friend classes and functions.

protected elements are just like the private, except they can be accessed by derived classes.

Note: By default, class members in C++ are **private**, unless specified otherwise.

Specifiers	Same Class	Derived Class	Outside Class
Public	Yes	Yes	Yes
Private	Yes	No	No
protected	Yes	Yes	No

Constructor in C++

- ❖ Special method that is implicitly invoked.
- ❖ Used to create an object (an instance of the class) and initialize it.
- ❖ Every time an object is created, at least one constructor is called.
- ❖ It is special member function having same name as class name and is used to initialize object.
- ❖ It is invoked/called at the time of object creation.
- ❖ It constructs value i.e. provide data for the object that is why it called constructor.
- ❖ Can have parameter list or argument list. Can never return any value (no even void).
- ❖ Normally declared as public.
- ❖ At the time of calling constructor, memory for the object is allocated in the memory.
- ❖ It calls a default constructor if there is no constructor available in the class.

Note: It is called constructor because it constructs the values at the time of object creation.

There can be two types of constructors in C++.

1. Default constructor (no-argument constructor)
2. Parameterized constructor.

1. Default constructor

- ❖ A constructor is called "Default Constructor" when it doesn't have any parameter.
- ❖ It is also called non-parameterized constructor.
- ❖ A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.
- ❖ Note: If we have not defined a constructor in our class, then the C++ compiler will automatically create a default constructor with an empty code and no parameters.
- ❖ Let's see the simple example of C++ default Constructor.

```
#include <iostream>
using namespace std;
class Employee
{
    public:
        Employee()
        {
            cout<<"Default Constructor Invoked/Called"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
```

```
        return 0;
    }
    /*
Output
Default Constructor Invoked/Called
Default Constructor Invoked/Called
*/
```

```
#include<iostream>
#include<string.h>
using namespace std;
class Student
{
    int Roll;
    char Name[25];
    float Marks;
public:
    Student()           //Default Constructor
    {
        Roll = 1;
        strcpy(Name,"Kumar");
        Marks = 78.42;
    }
    void Display()
    {
        cout<<"\n\tRoll : "<<Roll;
        cout<<"\n\tName : "<<Name;
        cout<<"\n\tMarks : "<<Marks;
    }
}; // end of class
int main()
{
    Student S;           //Creating Object

    S.Display();         //Displaying Student Details
    return 0;

}

/*
Output
Roll : 1
Name : Kumar
Marks : 78.42
*/
```

```
#include <string>
#include <iostream>
using namespace std;
class Classroom {
private:
int roomID;
int numberOfChairs;
char boardType; // C for chalk and M for marker
string multimedia;
string remarks;
public:
    Classroom();
    void setroomID(int);
    void setnumberOfChairs(int);
    void setboardType(char);
    void setmultimedia(string);
    void setremarks(string);
    int getroomID();
    int getnumberOfChairs();
    char getboardType();
    string getmultimedia();
    string getremarks();
    void display();
};
int main ()
{
    Classroom CR0;
    CR0.display();
    system("PAUSE");
    return 0;
}
//-----Constructors
Classroom::Classroom()
{
    this->roomID=0;
    this->numberOfChairs=0;
    this->boardType='M';
    this->multimedia="New";
    this->remarks="Default Constructor";
}
//-----Getter Functions
int Classroom::getroomID()
{
    return this->roomID;
}
int Classroom::getnumberOfChairs()
{
    return this->numberOfChairs;
}
char Classroom::getboardType()
```

```

{
    return (this->boardType);
}
string ClassRoom::getmultimedia()
{
    return this->multimedia;
}

string ClassRoom::getremarks()
{
    return this->remarks;
}
//-----Printing Functions
void ClassRoom::display()
{
    cout<<"Room ID \t: "<<this->getroomID()<<endl;
    cout<<"N.O. Chairs \t: "<<this->getnumberOfChairs()<<endl;
    cout<<"Board Type \t: "<<this->getboardType()<<endl;
    cout<<"Multimedia \t: "<<this->getmultimedia()<<endl;
    cout<<"Remarks \t: \n"<<this->getremarks()<<endl;
    cout<<"-----"<<endl;
}

/*
Room ID      : 0
N.O. Chairs  : 0
Board Type   : M
Multimedia   : New
Remarks     :
Default Constructor
-----
Press any key to continue . . .
*/

```

2. Parameterized Constructor

❖ A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.
- Used to initialize objects with different values.
- This is the preferred method to initialize member data.

❖ Let's see the simple example of C++ Parameterized Constructor.

```

#include <iostream>
using namespace std;
class Employee {
public:
    int id; //data member (also instance variable)
    string name; //data member(also instance variable)
    float salary;

    Employee(int i, string n, float s)
    {
        id = i;
        name = n;
        salary = s;
    }
void display()
{
    cout<<id<<" "<<name<<" "<<salary<<endl;
}
};
int main(void) {
    Employee e1 =Employee(101, "Ali", 890000); //creating an object of E
mployee
    Employee e2=Employee(102, "Saad", 59000);
    e1.display();
    e2.display();
    return 0;
}

/*
Output
101  Ali  890000
102  Saad  59000
*/

```

```

#include<iostream>
#include<string.h>
using namespace std;

class Student
{
    int Roll;
    char Name[25];
    float Marks;
}

```

```

        public:
        Student(int r,char nm[],float m)    //Parameterize Constructor
        {
            Roll = r;
            strcpy(Name,nm);
            Marks = m;
        }
void Display()
{
    cout<<"\n\tRoll : "<<Roll;
    cout<<"\n\tName : "<<Name;
    cout<<"\n\tMarks : "<<Marks;
}
};

int main()
{
    Student S(2,"Sumit",89.63);
    //Creating Object and passing values to Constructor

    S.Display();
    //Displaying Student Details
    return 0;
}

/*
Output
Roll : 2
Name : Ali
Marks : 89.63
*/

```

```

#include <string>
#include <iostream>
using namespace std;
class Classroom {
private:
int roomID;
int numberOfChairs;
char boardType; // C for chalk and M for marker
string multimedia;
string remarks;
public:
    Classroom(int id,int NOC,char,string mul, string rmks);
    void setroomID(int);

```

```

        void setnumberOfChairs(int);
        void setboardType(char);
        void setmultimedia(string);
        void setremarks(string);
        int getroomID();
        int getnumberOfChairs();
        char getboardType();
        string getmultimedia();
        string getremarks();
        void display();
};
int main ()
{
    Classroom CR(11,25,'M',"Repairing..","Remarks added from Main");
    CR.display();
    system("PAUSE");
    return 0;
}
//-----Constructors
Classroom::Classroom(int id,int NOC,char c,string mul, string remks)
{
    this->roomID=id;
    this->numberOfChairs=NOC;
    this->boardType=c;
    this->setmultimedia(mul);
    this->setremarks(remks+"\nConstructor with All(5)Parameters
(ID,NOC,BoardType,Multimedia,Remarks)");
}
//-----Setter Functions
void Classroom::setroomID(int id)
{
    this->roomID=id;
}
void Classroom::setnumberOfChairs(int NOC)
{
    this->numberOfChairs=NOC;
}
void Classroom::setboardType(char c)
{
    this->boardType=c;
}
void Classroom::setmultimedia(string str)
{
    this->multimedia=str;
}
void Classroom::setremarks(string str)
{
    this->remarks=str;
}
//-----Getter Functions
int Classroom::getroomID()

```



```

{
    return this->roomId;
}
int Classroom::getnumberOfChairs()
{
    return this->numberOfChairs;
}
char Classroom::getboardType()
{
    return (this->boardType);
}
string Classroom::getmultimedia()
{
    return this->multimedia;
}

string Classroom::getremarks()
{
    return this->remarks;
}
//-----Printing Functions
void Classroom::display()
{
    cout<<"Room ID \t: "<<this->getroomId()<<endl;
    cout<<"N.O. Chairs \t: "<<this->getnumberOfChairs()<<endl;
    cout<<"Board Type \t: "<<this->getboardType()<<endl;
    cout<<"Multimedia \t: "<<this->getmultimedia()<<endl;
    cout<<"Remarks \t: \n"<<this->getremarks()<<endl;
    cout<<"-----"<<endl;
}
/*
Room ID          : 11
N.O. Chairs      : 25
Board Type       : M
Multimedia       : Repairing..
Remarks         :
Remarks added from Main
Constructor with All(5)Parameters (ID,NOC,BoardType,Multimedia,Remarks)
-----
*/

```

The Default Copy Constructor

- ❖ We've seen two ways to initialize objects. A no-argument constructor can initialize data members to constant values, and a multi-argument constructor can initialize data members to values passed as arguments.
- ❖ Let's mention another way to initialize an object: you can initialize it with another object of the same type.

- ❖ Surprisingly, you don't need to create a special constructor for this; one is already built into all classes. It's called the default copy constructor. It's a one argument constructor whose argument is an object of the same class as the constructor.
- ❖ Initialization of an object through another object is called **copy constructor**.
- ❖ In other words, copying the values of one object into another object is called **copy constructor**.

Example of The Default Copy Constructor

```
#include<iostream>
#include<string.h>
using namespace std;

class Student
{
    int Roll;
    string Name;
    float Marks;

    public:
    Student(int r,string nm,float m)    //Parameterized Constructor
    {
        Roll = r;
        Name=nm;
        Marks = m;
    }

    // Student(Student &S)
    // {
    //     Roll=S.Roll;
    //     Name=S.Name;
    //     Marks =S.Marks;
    // }

    void Display()
    {
        cout<<"\n\tRoll : "<<Roll;
        cout<<"\n\tName : "<<Name;
        cout<<"\n\tMarks : "<<Marks;
    }
}; // end of class
```

```

int main()
{
    Student S1(2,"Ali",89.63);

    Student S2(S1);    //Copy S1 to S2
    Student S3=S1;    //copy S1 to S3

    cout<<"\n\tValues in object S1";
    S1.Display();

    cout<<"\n\tValues in object S2";
    S2.Display();

    cout<<"\n\tValues in object S3";
    S3.Display();
} // end of main() function

```

```

/*

```

Output

```

Values in object S1
Roll : 2
Name : Ali
Marks : 89.63
Values in object S2
Roll : 2
Name : Ali
Marks : 89.63
Values in object S3
Roll : 2
Name : Ali
Marks : 89.63
*/

```

- ❖ We initialize S1 to the value of "2,"Ali",89.63" using the three-argument constructor. Then we define two more objects of type Student Class, S2 and S3, initializing both to the value of S1.
- ❖ You might think this would require us to define a one-argument constructor, but initializing an object with another object of the same type is a special case. These definitions both use the default copy constructor.
- ❖ The object S2 is initialized in the statement

```

Student S2(S1);

```

- ❖ This causes the default copy constructor for the Student class to perform a member-by-member copy of S1 into S2.
- ❖ Surprisingly, a different format has exactly the same effect, causing S1 to be copied member-by-member into S3:

Student S3 = S1;

Constructor Overloading

- ❖ More than one constructor functions can be defined in one class. When more than one constructor functions are defined, each constructor is defined with a different set of parameters.
- ❖ Defining more than one constructor with different set of parameters is called constructor overloading.
- ❖ Constructor overloading is used to initialize different values to class objects.
- ❖ When a program that uses the constructor overloading is compiled, C++ compiler checks the number of parameters, their order and data types and marks them differently.
- ❖ When an object of the class is created, the corresponding constructor that matches the number of parameters of the object function is executed.
- ❖ In the following example two constructor functions are defined in the class “Sum”.

```
using namespace std;
#include<iostream>
class Sum
{
public:
    Sum(int l, int m, int n)
    {
        cout<<"Sum of three integer is= "<<(l+m+n)<<endl;
    }
    Sum(int l, int m)
    {
        cout<<"Sum of two integer is= "<<(l+m)<<endl;
    }
}; // end of class body

int main () {

    Sum s1=Sum(3,4,5);
    Sum s2=Sum(2,4);
    //Sum s1(3,4,5), s2(2,4);
```

```

    return 0;
}

```

```

/*

```

Output

```

Sum of three integer is= 12

```

```

Sum of two integer is= 6

```

```

*/

```

- ❖ When the above program is executed, the object s1 is created first and then the Sum constructor function that has only three integer type parameters is executed.
- ❖ Then the s2 object is created. It has two parameters of integer type, So the constructor function that has two arguments of integer type is executed.

Write a program to define two constructors to find out the maximum values.

```

using namespace std;
#include<iostream>
class Find
{
    private:
        int max;
    public:
        Find(int x, int y, int z)
        {
            if (x>y)
            {
                if(x>z)
                {
                    max=x;
                }
                else
                {
                    max=z;
                }
            }
            else if(y>z)
                max=y;
            else
                max=z;
            cout<<"Maximum between three numbers is= "<<max<<endl;
        }
        Find(int x, int y)
        {

```

```

        if (x>y)
        {
            max=x;
        }
        else
        {
            max=y;
        }
        cout<<"Maximum between two numbers is= "<<max<<endl;
    }
}; // end of class body

int main () {

    int a=9, b=56, c=67;
    Find f1=Find(a,b,c);
    Find f2=Find(a,b);
    //Find f1(a,b,c), f2(a,b);

    return 0;
}

/*
Output
Maximum between three numbers is = 67
Maximum between two numbers is = 56
*/

```

```

//Given example demonstrates the concept of overloaded constructor
#include <string>
#include <iostream>
using namespace std;
class ClassRoom {
private:
    int roomID;
    int numberOfChairs;
    char boardType; // C for chalk and M for marker
    string multimedia;
    string remarks;
public:
    ClassRoom();
    ClassRoom(int id);
    ClassRoom(int id,int NOC);
    ClassRoom(int id,int NOC,char c);

```

```
ClassRoom(int id,int NOC,char,string mul);
ClassRoom(int id,int NOC,char,string mul, string rmks);

void setroomID(int);
    void setnumberOfChairs(int);
    void setboardType(char);
    void setmultimedia(string);
    void setremarks(string);
    int getroomID();
    int getnumberOfChairs();
    char getboardType();
    string getmultimedia();
    string getremarks();
    void display();

};
int main ()
{
ClassRoom CR0,CR1(1),CR2(2,30),CR3(3,35,'M'),
CR4(4,40,'M',"Repairing.."),CR5(5,40,'M',"Repairing..","Remarks added
from Main");
CR0.display();
CR1.display();
CR2.display();
CR3.display();
CR4.display();
CR5.display();
return 0;
}
//-----Constructors
ClassRoom::ClassRoom()
{
    this->roomID=0;
    this->numberOfChairs=0;
    this->boardType='M';
    this->multimedia="New";
    this->remarks="Default Constructor";
}
ClassRoom::ClassRoom(int id)
{
this->roomID=id;
this->setnumberOfChairs(0);
this->setboardType('M');
this->setmultimedia("New");
this->remarks="Constructor with 1-Parameter (ID) ";
}
```

```
ClassRoom::ClassRoom(int id,int NOC)
{
this->roomID=id;
this->numberOfChairs=NOC;
this->setboardType('M');
this->setmultimedia("New");
this->remarks="Constructor with 2-Parameters (ID,NOC) ";
}
ClassRoom::ClassRoom(int id,int NOC,char c)
{
this->roomID=id;
this->numberOfChairs=NOC;
this->boardType=c;
this->setmultimedia("New");
this->remarks="Constructor with 3-Parameters (ID,NOC,BoardType) ";
}
ClassRoom::ClassRoom(int id,int NOC,char c,string mul)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->setremarks("Constructor with 4-
Parameters (ID,NOC,BoardType,Multimedia)");
}
ClassRoom::ClassRoom(int id,int NOC,char c,string mul, string remks)
{
this->setroomID(id); //this->roomID=id;
this->setnumberOfChairs(NOC); //this->numberOfChairs=NOC;
this->setboardType(c); //this->boardType=c;
this->setmultimedia(mul); //this->multimedia=str;
this->
>setremarks(remks+"\nConstructor with All(5)Parameters (ID,NOC,BoardTy
pe,Multimedia,Remarks)");
}

//-----Setter Functions
void ClassRoom::setroomID(int id)
{
    this->roomID=id;
}
void ClassRoom::setnumberOfChairs(int NOC)
{
    this->numberOfChairs=NOC;
}
```



```
void Classroom::setboardType(char c)
{
    this->boardType=c;
}
void Classroom::setmultimedia(string str)
{
    this->multimedia=str;
}
void Classroom::setremarks(string str)
{
    this->remarks=str;
}
//-----Getter Functions
int Classroom::getroomID()
{
    return this->roomID;
}
int Classroom::getnumberOfChairs()
{
    return this->numberOfChairs;
}
char Classroom::getboardType()
{
    return (this->boardType);
}
string Classroom::getmultimedia()
{
    return this->multimedia;
}

string Classroom::getremarks()
{
    return this->remarks;
}
//-----Printing Functions
void Classroom::display()
{
    cout<<"Room ID \t: "<<this->getroomID()<<endl;
    cout<<"N.O. Chairs \t: "<<this->getnumberOfChairs()<<endl;
    cout<<"Board Type \t: "<<this->getboardType()<<endl;
    cout<<"Multimedia \t: "<<this->getmultimedia()<<endl;
    cout<<"Remarks \t: \n"<<this->getremarks()<<endl;
    cout<<"-----"<<endl;
}
```

```
/*
Output:
Room ID : 0
N.O. Chairs : 0
Board Type : M
Multimedia : New
Remarks :
Default Constructor
-----
Room ID : 1
N.O. Chairs : 0
Board Type : M
Multimedia : New
Remarks :
Constructor with 1-Parameter (ID)
-----
Room ID : 2
N.O. Chairs : 30
Board Type : M
Multimedia : New
Remarks :
Constructor with 2-Parameters (ID,NOC)
-----
Room ID : 3
N.O. Chairs : 35
Board Type : M
Multimedia : New
Remarks :
Constructor with 3-Parameters (ID,NOC,BoardType)
-----
Room ID : 4
N.O. Chairs : 40
Board Type : M
Multimedia : Repairing..
Remarks :
Constructor with 4-Parameters (ID,NOC,BoardType,Multimedia)
-----
Room ID : 5
N.O. Chairs : 40
Board Type : M
Multimedia : Repairing..
Remarks :
Remarks added from Main
Constructor with All(5)Parameters (ID,NOC,BoardType,Multimedia,Remarks)
-----
Press any key to continue . . .
*/
```

Defining Constructor Outside Class

```
class class_name {
public:
    //Constructor declaration
    class_name();
    //... other Variables & Functions
}; // Class body ends

// Constructor definition outside Class
class_name::class_name()
{
    // Constructor code
}
```

Defining Constructor Outside Class Example

```
using namespace std;
#include<iostream>
class Sum
{
    //Constructor declaration
public:
    Sum(int l, int m, int n);
    Sum(int l, int m);
}; // end of class body
int main () {
    Sum s1=Sum(3,4,5);
    Sum s2=Sum(2,4);
    //Sum s1(3,4,5), s2(2,4);

    return 0;
} //end of main() function
// Constructor definition outside Class
Sum::Sum(int l, int m, int n)
{
    cout<<"Sum of three integer is= "<<(l+m+n)<<endl;
}
Sum::Sum(int l, int m)
{
    cout<<"Sum of two integer is= "<<(l+m)<<endl;
}
/*
Output
```

*/

References

<https://beginnersbook.com/2017/08/cpp-data-types/>

http://www.cplusplus.com/doc/tutorial/basic_io/

<https://www.w3schools.com/cpp/default.asp>

<https://www.javatpoint.com/cpp-tutorial>

<https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp>

<https://www.programiz.com/>

<https://ecomputernotes.com/cpp/>