FAST

National University of Computer and Emerging Sciences Peshawar

OOP Lab # 09

C++ (Classes and Objects)

Computer Instructor: Muhammad Abdullah Orakzai

DEPARTMENT OF COMPUTER SCIENCE

Programming

الذی علم بالقلم۔ علم الانسان ما لم یعلم۔

Contents

1) Access Specifiers/Modifiers in C++
2) Constructor
3) Types of Constructor (Default Constructor and Parametrized Constructor)
4) Default Copy Constructor
5) Constructor Overloading
6) Defining Constructor Outside Class
7) Destructors
8) C++ Objects and Functions
9) Passing objects as arguments to function
10) Returning objects from function
11) Arrays as Class Members in C++

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

Access Specifiers/Modifiers in C++…

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;
        }

};
```

Example 1: Public Access Specifier…

```
// 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

Example 1: Public Access Specifier…

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
```

Example 1: Private Access Specifier...

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

    // 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; ^
```

Example 1: Private Access Specifier…

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;
```

Example 2: Private Access Specifier...

```
// 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
```

Example 2: Private Access Specifier...

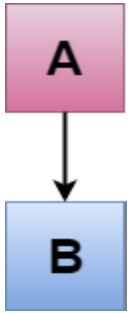
```
// main function
int main()
{
    // creating object of the class
    Circle obj;

    // 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 it's 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
```

Example 1: Protected Access Specifier…

```
// 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
```

Example 1: Protected Access Specifier...

```
// 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.

Summary: public, private, and protected

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.

Constructor in C++…

- 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.

Constructor in C++…

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.

1. 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

1. Default constructor…

```
#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;
            }
```

1. Default constructor…

```
 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

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.

2. 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;
         }
```

2. Parameterized Constructor…

```
void display()
        {
            cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};
int main(void) {
  Employee e1 =Employee(101, "Ali", 890000); //creating an object of Employee

    Employee e2=Employee(102, "Saad", 59000);
    e1.display();
    e2.display();
    return 0;
}
```

Output:

101 Ali 890000
102 Saad 59000

2. Parameterized Constructor…

```
#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;
        }
```

2. Parameterized Constructor…

```
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

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.

The Default Copy 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;
        }
```

Example of The Default Copy Constructor…

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

Example of The Default Copy Constructor…

```
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

Example of The Default Copy Constructor…

- 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);
```

Example of The Default Copy Constructor…

- 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.

Constructor Overloading…

- 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;
   }
```

Constructor Overloading…

```
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

Constructor Overloading…

- 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.

Constructor Overloading…

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

```
using namespace std;
#include<iostream>
class Find
{
   private:
   int max;
```

Constructor Overloading…

```
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;
   }
```

Constructor Overloading…

```
 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
```

Constructor Overloading…

```
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

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
```

Defining Constructor Outside Class Example…

```
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
```

Defining Constructor Outside Class Example…

```
// 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:
Maximum between three numbers is = 67
Maximum between two numbers is = 56

Destructors

- When an object is destroyed, a special member function of that class is executed automatically. This member functions is called **destructor function** or **destructor**.
- The destructor function has the same name as the name of a class but a tilde sign (~) is written before its name. It is executed automatically when an object comes to end of its life.
- Like constructors, destructor do not return any value. They also do not take any arguments.
- **For example**, a local object is destroyed when all the statements of the function in which it is declared are executed. So at the end of the function, the destructor function is executed.

Destructors…

- Similarly, global objects (objects that are declared before main function) or static objects are destroyed at the end of main function. The life time of these objects end when the program execution ends.
- So at the end of program the destructor function is executed. The destructor functions are commonly used to free the memory that was allocated for objects.
- Constructor is invoked automatically when the object created.
- Destructor is invoked when the object goes out of scope. In other words, Destructor is invoked, when compiler comes out from the function where an object is created.
- The following example explains the concept of constructors and destructors.

Destructors...

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

class Prog
{
    public:
    Prog()
    {
        cout<<"This is constructor function "<<endl;
    }
```

Destructors...

```
    ~Prog()
    {
        cout<<"This is destructor function "<<endl;
    }

}; // end of class body

int main () {

    Prog x;
    int a, b;
    a=10;
    b=20;
    cout<<"Sum of two numbers is = "<<(a+b)<<endl;

   return 0;
}
```

Output:

This is constructor function
Sum of two numbers is = 30
This is destructor function

C++ Objects and Functions

- In this tutorial, we will learn to pass objects to a function and return an object from a function in C++ programming.
- In C++ programming, we can pass objects to a function in a similar manner as passing regular arguments.

Passing Objects as Arguments to Function

Objects can also be passed as arguments to member functions. When an objects is passed as an argument to a member function:

only the name of the object is written in the argument.

The number of parameters and their types must be defined in the member function which the object is to be passed. The objects that are passed are treated local for the member functions and are destroyed when the control returns to the calling function.

Example 1: C++ Pass Objects to Function

```
using namespace std;
#include<iostream>
class Test
{
    private:
    char name[20];

    public:
    void get()
    {
        cout<<"Enter your name: ";
        cin.get(name, 20);
    }
```

Example 1: C++ Pass Objects to Function...

```
  void print(Test s)
    {
        cout<<"Name is: "<<s.name<<endl;
    }

}; // end of class body

int main () {

    Test test1,test2;
    test1.get(); // calling get() function for object initialization

    test2.print(test1);  //Passing object as argument to function

   return 0;
}
```

Output:

Enter your name: Nouman Yousaf
Name is: Nouman Yousaf

Example 1: C++ Pass Objects to Function…

- In the above program, the class “**Test**” has one data member “**name**” of string type and two member functions “**get()**” and “**print()**”. The **print()** function has parameter of class **Test** type.
- The objects **test1** and **test2** are declared of class **Test**. The member function gets the name in object test1, and store it into the data member “**name**”. The member function “**print()**” for objects “**test2**” is called by passing argument of object “**test1**”. When the control shifts to member function “**print()**”, a copy of “**test1**” is created as a local object in the print function with name “**s**”.

Example 2: C++ Pass Objects to Function

```
// C++ program to calculate the average marks of two students
#include <iostream>
using namespace std;

class Student {

   public:
    double marks;

    // constructor to initialize marks
    Student(double m) {
        marks = m;
    }
};  // class body ends
```

Example 2: C++ Pass Objects to Function...

```
// function that has objects as parameters
void calculateAverage(Student s1, Student s2) {

    // calculate the average of marks of s1 and s2
    double average = (s1.marks + s2.marks) / 2;

   cout << "Average Marks = " << average << endl;

}
int main() {
    Student student1(88.0), student2(56.0);

  // pass the objects as arguments
   calculateAverage(student1, student2);

    return 0;
}
```

Output:
Average Marks = 72

Example 2: C++ Pass Objects to Function...

- Here, we have passed two Student objects **student1** and **student2** as arguments to the **calculateAverage()** function.

```
#include<iostream>

class Student {...};

void calculateAverage(Student s1, Student s2) {
    // code
}

int main() {
    ... ...
    calculateAverage(student1, student2);
    ... ...
}
```

Example 3: C++ Return Object from a Function

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

class Student {
   public:
    double marks1, marks2;
};  //class body ends
```

Example 3: C++ Return Object from a Function…

```
// function that returns object of Student
Student createStudent() {
    Student student;

    // Initialize member variables of Student
    student.marks1 = 96.5;
    student.marks2 = 75.0;

    // print member variables of Student
    cout << "Marks 1 = " << student.marks1 << endl;
    cout << "Marks 2 = " << student.marks2 << endl;

    return student;
}
```

Example 3: C++ Return Object from a Function…

```
int main() {

    Student student1;

    // Call function

    student1 = createStudent();



    return 0;

}
```

Output:
Marks 1 = 96.5
Marks 2 = 75

Arrays as Class Members in C++

- Arrays can be declared as the members of a class. The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.
- Example: A program to demonstrate the concept of arrays as class members
- The output of the program is

Arrays as Class Members in C++...

```
#include<iostream>
using namespace std;
const int size=5;
class Student
{
  int roll_no;
  int marks[size];

  public:
    void getdata ();
    void tot_marks ();
};  // end of clas body
```

Arrays as Class Members in C++...

```
int main()
{
    Student s1;
    s1.getdata() ;
    s1.tot_marks() ;
    return 0;
} //ends of main() function
```

Arrays as Class Members in C++...

```
//function definitions
void Student :: getdata ()
{
      cout<<"\nEnter roll no: ";
      cin>>roll_no;
      for(int i=0; i<size; i++)
      {
        cout<<"Enter marks in subject"<<(i+1)<<": ";
        cin>>marks[i] ;
      }
}
```

Arrays as Class Members in C++...

```
//calculating total marks
void Student :: tot_marks ()
{
      int total=0;
      for(int i=0; i<size; i++)
      {
          total=total+marks[i];
      }
      cout<<"\nTotal marks "<<total;
}
```

Output:

Enter roll no: 333
Enter marks in subject1: 56
Enter marks in subject2: 88
Enter marks in subject3: 77
Enter marks in subject4: 66
Enter marks in subject5: 88

Total marks 375

Arrays as Class Members in C++...

In this example, an array marks is declared as a private member of the class student for storing a student's marks in five subjects. The member function tot_marks () calculates the total marks of all the subjects and displays the value.

Similar to other data members of a class, the memory space for an array is allocated when an object of the class is declared. In addition, different objects of the class have their own copy of the array. Note that the elements of the array occupy contiguous memory locations along with other data members of the object. For instance, when an object s1 of the class student is declared, the memory space is allocated for both rollno and marks.

Ways to initialize object

There are 3 ways to initialize object in C++.

- By directly accessing data members of class using object
- By member functions of the class
- By constructors of class

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/

THANK YOU