

FAST

National University of Computer and Emerging Sciences Peshawar

OOP Lab # 12

C++ (Classes and Objects)

Computer Instructor: Engr. Khuram Shahzad

DEPARTMENT OF COMPUTER SCIENCE



Programming



الذى علم بالقلم. علم الانسان ما لم يعلم.



OOP Lab-12: 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++



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...

Program.cpp

```
1 #include <iostream>
2 using namespace std;
3 class Employee {
4     public:
5         int id; //data member (also instance variable)
6         string name; //data member(also instance variable)
7         float salary;
8
9         Employee(int i, string n, float s)
10        {
11            id = i;
12            name = n;
13            salary = s;
14        }
15        void display()
16        {
17            cout<<id<<" "<<name<<" "<<salary<<endl;
18        }
19 };
20 int main(void) {
21     Employee e1 =Employee(101, "Ali", 890000); //creating an object of Employee
22     Employee e2=Employee(102, "Saad", 59000);
23     e1.display();
24     e2.display();
25     return 0;
26 }
```

Output:

```
101 Ali 890000
102 Saad 59000
```



2. Parameterized Constructor...

```
1 #include<iostream>
2 #include<string.h>
3 using namespace std;
4 class Student
5 {
6     int Roll;
7     char Name[25];
8     float Marks;
9     public:
10    Student(int r,char nm[],float m)           //Parameterize Constructor
11    {
12        Roll = r;
13        strcpy(Name,nm);
14        Marks = m;
15    }
16    void Display()
17    {
18        cout<<"\n\tRoll : "<<Roll;
19        cout<<"\n\tName : "<<Name;
20        cout<<"\n\tMarks : "<<Marks;
21    }
22 };
23 int main()
24 {
25     Student S(2,"Sumit",89.63);
26     //Creating Object and passing values to Constructor
27     S.Display();
28     //Displaying Student Details
29     return 0;
30 }
```

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...

```

1 #include<iostream>
2 #include<string.h>
3 using namespace std;
4 class Student
5 {
6     int Roll;
7     string Name;
8     float Marks;
9     public:
10    Student(int r,string nm,float m)  //Parameterized Constructor
11    {
12        Roll = r;
13        Name=nm;
14        Marks = m;
15    }
16    void Display()
17    {
18        cout<<"\n\tRoll : "<<Roll;
19        cout<<"\n\tName : "<<Name;
20        cout<<"\n\tMarks : "<<Marks;
21    }
22 }; // end of class
23 int main()
24 {
25     Student S1(2,"Ali",89.63);
26     Student S2(S1);  //Copy S1 to S2
27     cout<<"\n\tValues in object S1";
28     S1.Display();
29     cout<<"\n\tValues in object S2";
30     S2.Display();
31     cout<<"\n\tValues in object S3";
32     S3.Display();
33 } // end of main() function
34

```

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”** .

Constructor Overloading...

```
[*] Program.cpp
1  #include<iostream>
2  using namespace std;
3  class Sum
4  {
5  public:
6      Sum(int l, int m, int n)
7      {
8          cout<<"Sum of three integer is= "<<(l+m+n)<<endl;
9      }
10     Sum(int l, int m)
11     {
12         cout<<"Sum of two integer is= "<<(l+m)<<endl;
13     }
14 }; // end of class body
15 int main () {
16     Sum s1=Sum(3,4,5);
17     Sum s2=Sum(2,4);
18     //Sum s1(3,4,5), s2(2,4);
19
20     return 0;
21 }
```

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.
- One constructor will give the max form 2 input parameter
- Second constructor will give the max form 3 input parameter

Constructor Overloading...

```
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



Defining Constructor Outside Class Example...

```
1 using namespace std;
2 #include<iostream>
3 class Sum
4 {
5     //Constructor declaration
6     public:
7     Sum(int l, int m, int n);
8     Sum(int l, int m);
9 }; // end of class body
10 int main () {
11     Sum s1=Sum(3,4,5);
12     Sum s2=Sum(2,4);
13     //Sum s1(3,4,5), s2(2,4);
14
15     return 0;
16 } //end of main() function
17 // Constructor definition outside Class
18 Sum::Sum(int l, int m, int n)
19 {
20     cout<<"Sum of three integer is= "<<(l+m+n)<<endl;
21 }
22
23 Sum::Sum(int l, int m)
24 {
25     cout<<"Sum of two integer is= "<<(l+m)<<endl;
26 }
```

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 function 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, destructors 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...

```
Program.cpp
1 using namespace std;
2 #include<iostream>
3 class Prog
4 {
5     public:
6     Prog()
7     {
8         cout<<"This is constructor function "<<endl;
9     }
10    ~Prog()
11    {
12        cout<<"This is destructor function "<<endl;
13    }
14 }; // end of class body
15
16 int main () {
17     Prog x;
18     int a, b;
19     a=10;
20     b=20;
21     cout<<"Sum of two numbers is = "<<(a+b)<<endl;
22     return 0;
23 }
```

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 object 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);
    }
    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...

Program.cpp

```
1 // C++ program to calculate the average marks of two students
2 #include <iostream>
3 using namespace std;
4 class Student {
5     public:
6         double marks;
7         // constructor to initialize marks
8         Student(double m) {
9             marks = m;
10        }
11 }; // class body ends
12 // function that has objects as parameters
13 void calculateAverage(Student s1, Student s2) {
14     // calculate the average of marks of s1 and s2
15     double average = (s1.marks + s2.marks) / 2;
16     cout << "Average Marks = " << average << endl;
17 }
18 int main() {
19     Student student1(88.0), student2(56.0);
20     // pass the objects as arguments
21     calculateAverage(student1, student2);
22     return 0;
23 }
```

Output:

Average Marks = 72

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

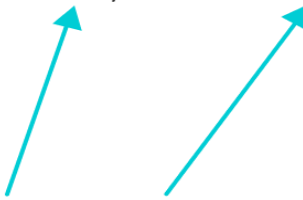
- Here, we have passed two Student objects **student1** and **student2** as arguments to the **calculateAverage()** function.

```
#include<iostream>

class Student {...};

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

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



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

```
Program.cpp
1 #include <iostream>
2 using namespace std;
3 class Student {
4     public:
5         double marks1, marks2;
6 }; //class body ends
7 // function that returns object of Student
8 Student createStudent() {
9     Student student;
10    // Initialize member variables of Student
11    student.marks1 = 96.5;
12    student.marks2 = 75.0;
13    // print member variables of Student
14    cout << "Marks 1 = " << student.marks1 << endl;
15    cout << "Marks 2 = " << student.marks2 << endl;
16    return student;
17 }
18 int main() {
19     Student student1; // Call function
20     student1 = createStudent();
21     return 0;
22 }
```

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



Arrays as Class Members in C++...

```
2 using namespace std;
3 const int size=5;
4 class Student
5 {
6     int roll_no;
7     int marks[size];
8     public:
9         void getdata ();
10        void tot_marks ();
11}; // end of clas body
12 int main()
13 {
14     Student s1;
15     s1.getdata() ;
16     s1.tot_marks() ;
17     return 0;
18 } //ends of main() function
19 //function definitions
20 void Student :: getdata ()
21 {
22     cout<<"\nEnter roll no: ";
23     cin>>roll_no;
24     for(int i=0; i<size; i++)
25     {
26         cout<<"Enter marks in subject"<<(i+1)<<": ";
27         cin>>marks[i] ;
28     }
29 }
```

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

