LAB NO. 06

CONSTRUCTORS AND DESTRUCTORS IN C++

Lab Objectives

Following are the lab objectives:

1. Constructors
    - Default constructor
    - Parameterize constructor
    - Copy constructor
2. Destructors

Instructions

- This is individual Lab work/task.
- Complete this lab work within lab timing.
- Discussion with peers is not allowed.
- Copy paste from Internet will give you **negative marks**.
- Lab work is divided into small tasks, complete all tasks sequentially.

1. Constructors and Destructors In C++

1.1 Constructor

A Constructor is a member function of a class. It is mainly used to initialize the objects of the class. It has the same name as the class. When an object is created, the constructor is automatically called. It is a special kind of member function of a class.

Difference between Constructor and Other Member Functions:

i. The Constructor has the same name as the class name.
ii. The Constructor is called when an object of the class is created.
iii. A Constructor does not have a return type.
iv. When a constructor is not specified, the compiler generates a default constructor which does nothing.

There are 3 types of constructors:

1. Default Constructor
2. Parameterized Constructor
3. Copy constructor

A constructor can also be defined in the private section of a class.

Default Constructor

A Default constructor is a type of constructor which doesn't take any argument and has no parameters.

Example 1

```
#include <iostream>
using namespace std;
class sum {
public:
int y, z;
sum()
{
y = 7;
z = 13;
}
};
int main()
{
```

```
sum a;
cout <<"the sum is: "<< a.y+a.z;
return 0;
}
```

Example 2

```
#include <iostream>
using namespace std;
class Line {
  public:
    void setLength( double len );
    double getLength( void );
    Line();  // This is the constructor
  private:
    double length;
};
// Member functions definitions including constructor
Line::Line(void) {
  cout << "Object is being created" << endl;
}
void Line::setLength( double len ) {
  length = len;
}
double Line::getLength( void ) {
  return length;
}
// Main function for the program
int main() {
  Line line;
  // set line length
  line.setLength(6.0);
  cout << "Length of line : " << line.getLength() <<endl;

  return 0;
}
```

Parameterized Constructor

Passing of parameters to the constructor is possible. This is done to initialize the value using these passed parameters. This type of constructor is called a parameterized constructor.

The constructor is defined as follows:

```
test(int x1)
{
x = x1;
}
```

There is a parameter that is passed to the constructor. The value is passed when the object is created in the main function as shown below.

```
test t(10);
```

Inside the main function, we create an object of class test and pass the value of the variable.

Example 3

```
#include <iostream>
using namespace std;
class test {
public:
int x;
test(int x1)
{
x = x1;
}
int getX()
{
return x;
}
};
int main()
{
test a(10);
cout << "a.x = " << a.getX() ;
return 0;
}
```

Example 4

```
#include <iostream>
```

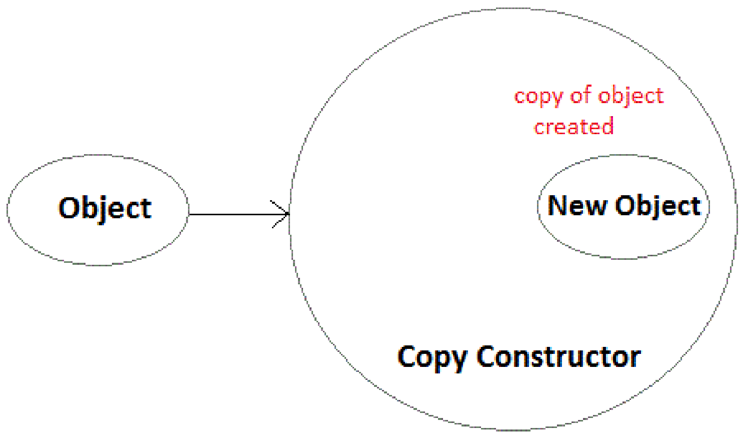
```
using namespace std;
class Line {
  public:
    void setLength( double len );
    double getLength( void );
    Line(double len);  // This is the constructor
  private:
    double length;
};
// Member functions definitions including constructor
Line::Line( double len) {
  cout << "Object is being created, length = " << len << endl;
  length = len;
}
void Line::setLength( double len ) {
  length = len;
}
double Line::getLength( void ) {
  return length;
}
// Main function for the program
int main() {
  Line line(10.0);
  // get initially set length.
  cout << "Length of line : " << line.getLength() <<endl;
  // set line length again
  line.setLength(6.0);
  cout << "Length of line : " << line.getLength() <<endl;
  return 0;
}
```

Copy Constructor

A Copy Constructor is a Constructor which initializes an object of a class using another object of the same class.

Syntax of Copy Constructor

```
Classname(const classname & objectname)
{
   . . . .
}
```



Example 5

```
#include<iostream>
using namespace std;
class test
{
private:
int x;
public:
test(int x1)
{
x = x1;
}
test(test &t2)
{
x = t2.x;
}
int getX()
{
return x;
}
};
int main()
{
test t1(7); // Normal constructor is called here
test t2 = t1; // Copy constructor is called here
cout << "t1.x = " << t1.getX();
cout << "nt2.x = " << t2.getX();
return 0;
}
```

2. Constructor Overloading in C++

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading and is quite similar to function overloading.

- Overloaded constructors essentially have the same name (name of the class) and different number of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Example 6

```
#include <iostream>
using namespace std;
class construct
{
public:
        float area;
        // Constructor with no parameters
        construct()
        {
                area = 0;
        }
        // Constructor with two parameters
        construct(int a, int b)
        {
                area = a * b;
        }
        void disp()
        {
                cout<< area<< endl;
        }
};
int main()
{
        construct o;
        construct o2( 10, 20);
        o.disp();
        o2.disp();
        return 0;
}
```

Destructors in C++

Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.

The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a tilde ~ sign as prefix to it.

```
class A
{
  public:
  // defining destructor for class
  ~A()
  {
    // statement
  }
};
```

Example to see how Constructor and Destructor are called

Below we have a simple class A with a constructor and destructor. We will create object of the class and see when a constructor is called and when a destructor gets called.

```
class A
{
   // constructor
   A()
   {
      cout << "Constructor called";
   }

   // destructor
   ~A()
   {
      cout << "Destructor called";
   }
};
int main()
{
   A obj1;   // Constructor Called
   int x = 1
   if(x)
   {
      A obj2;  // Constructor Called
   }   // Destructor Called for obj2
} //  Destructor called for obj1
```

```
Constructor called
Constructor called
Destructor called
Destructor called
```

Lab Tasks

Q1). Write a program to print the names of employee by creating an Employee class. If no name is passed while creating an object of the Employee class, then the name should be "Unknown", otherwise the name should be equal to the String value passed while creating the object of the Employee class.

Q2). Suppose you have an account in HBL Bank with an initial amount of 10k and you have to add some more amount to it. Create a class 'AddAmount' with a data member named 'amount' with an initial value of 10k. Now make two constructors of this class as follows:

1. without any parameter - no amount will be added to the account.
2. having a parameter which is the amount that will be added to the account.

Create an object of the 'AddAmount' class and display the final amount in the account.

Home Activity

Q.3. Declare a class of GPAin which user can store his/her total marks, obtained marks, Total marks in each course, Obtained marks in every course, Course name, Credit hrs and number of courses. Perform the following task using constructors:

1. Calculate gpa obtained in each course when One course marks are passed.
2. Calculate overall gpa when total marks and number of courses are passed.
3. Define destructors in the GPA class.

Q4. Declare a class **Area** which perform the following tasks using constructor chaining:

1. Calculate area of rectangle when length and width are passed.
2. Calculate area of circle when radius is passed.
3. Calculate area of cube when length, width and breadth is passed.
4. Define destructor at the end of Area class.