CECOS UNIVERSITY

Lecture 02

Object Oriented Programming

Arshi Parvaiz

Constructors

- A constructor is a member function that has the same name as the class.
- A constructor does not return anything.
- A constructor is automatically called when the object is created in memory, or instantiated.
- How ever its constructor can receive arguments just like other functions do.
- Constructor is a function that doesn’t need to be called.
- It is helpful to think of constructors as initialization routines.
- They are very useful for initializing member variables or performing other setup operations.

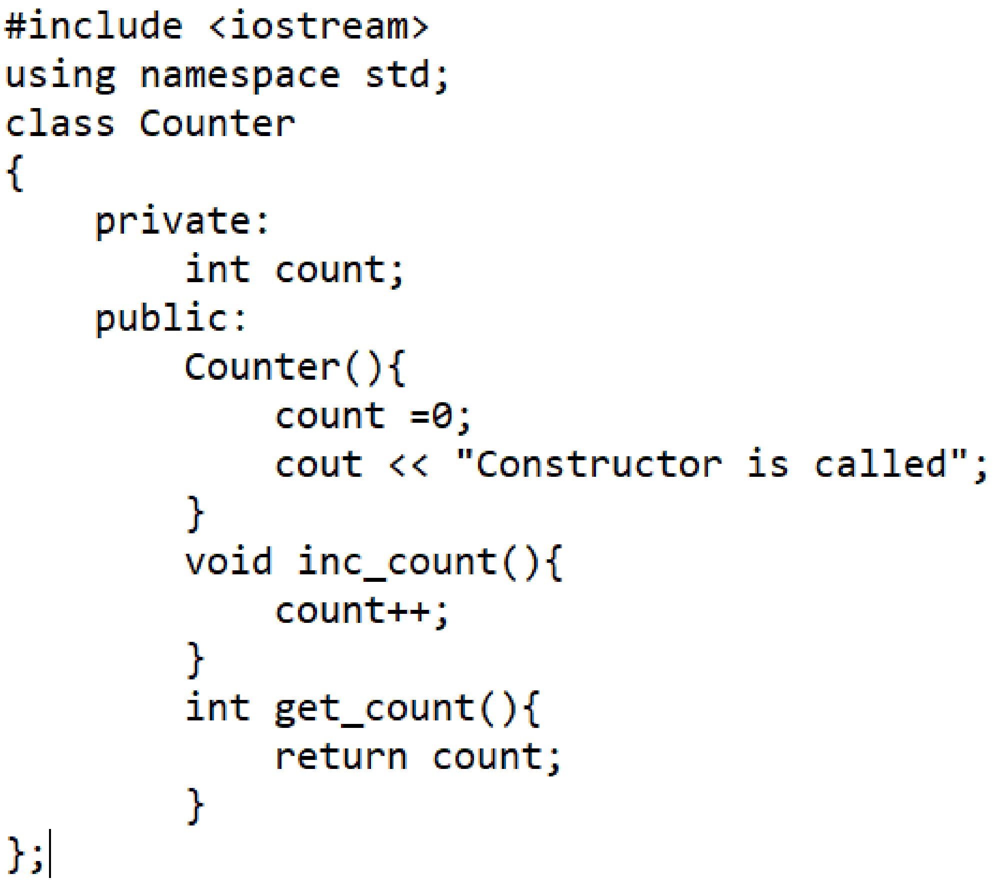
Constructors

- We can have multiple constructors in a class.
- If a programmer doesn’t include constructor in program, the compiler provides a default constructor, this constructor does not have parameters.
- Types of constructors
  - Default Constructor
  - Parameterized Constructor
  - Copy Constructor

Default Constructor

- Any constructor that takes no arguments is called a default constructor
- Default constructor can be set using two methods
  - As mentioned earlier the compiler implicitly creates a default constructor if not explicitly defined.
  - You explicitly define a constructor that takes no arguments (0 argument constructor)

Code Example

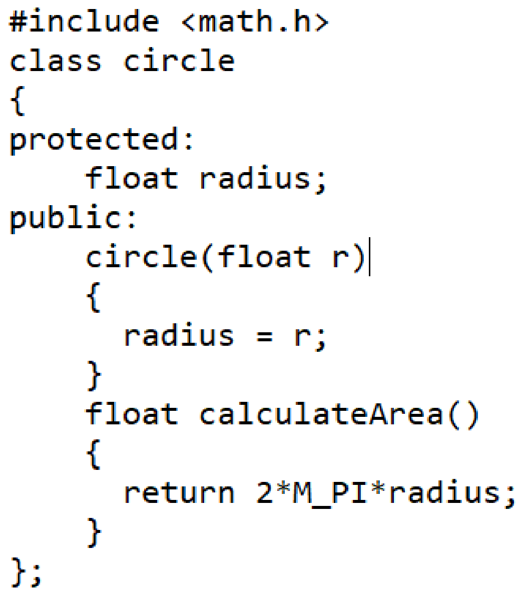


```
#include <iostream>
using namespace std;
class Counter
{
    private:
        int count;
    public:
        Counter(){
            count =0;
            cout << "Constructor is called";
        }
        void inc_count(){
            count++;
        }
        int get_count(){
            return count;
        }
};
```

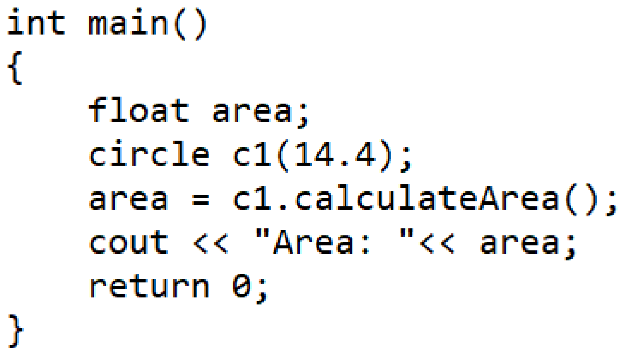
```
int main()
{
    Counter c1, c2;
    cout << “\nc1=” << c1.get_count();
    cout << “\nc2=” << c2.get_count();
    c1.inc_count();
    c2.inc_count();
    c2.inc_count();
    cout << “\nc1=” << c1.get_count();
    cout << “\nc2=” << c2.get_count();
    cout << endl;
    return 0;
}
```

Parameterized Constructor

- A parameterized constructor takes one or more arguments.
- **Code Example:**

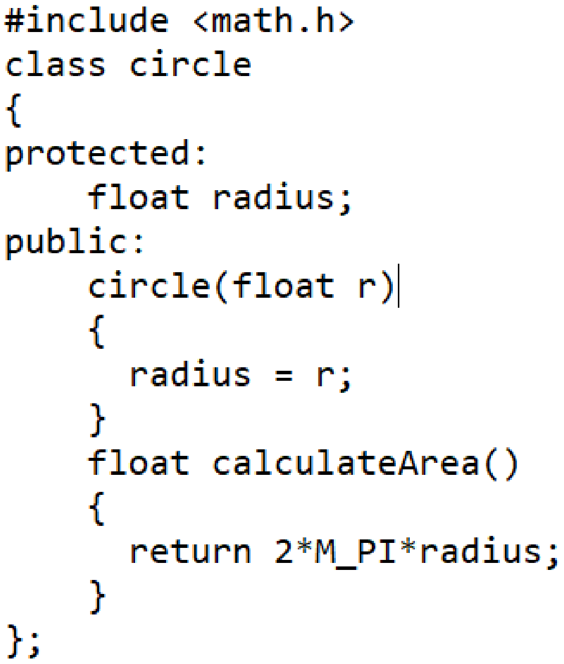


```
#include <math.h>
class circle
{
protected:
    float radius;
public:
    circle(float r)
    {
      radius = r;
    }
    float calculateArea()
    {
      return 2*M_PI*radius;
    }
};
```

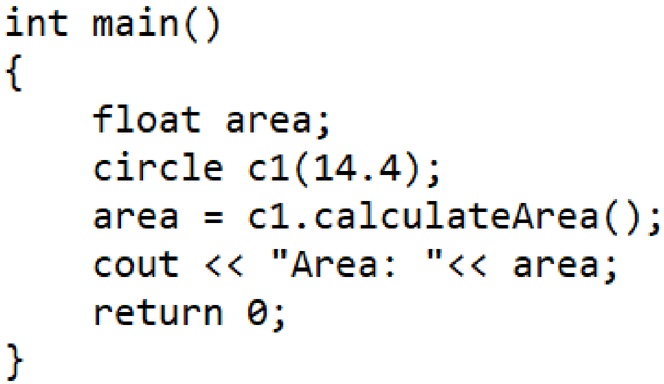


```
int main()
{
    float area;
    circle c1(14.4);
    area = c1.calculateArea();
    cout << "Area: "<< area;
    return 0;
}
```

Code Example



```
#include <math.h>
class circle
{
protected:
    float radius;
public:
    circle(float r)
    {
      radius = r;
    }
    float calculateArea()
    {
      return 2*M_PI*radius;
    }
};
```

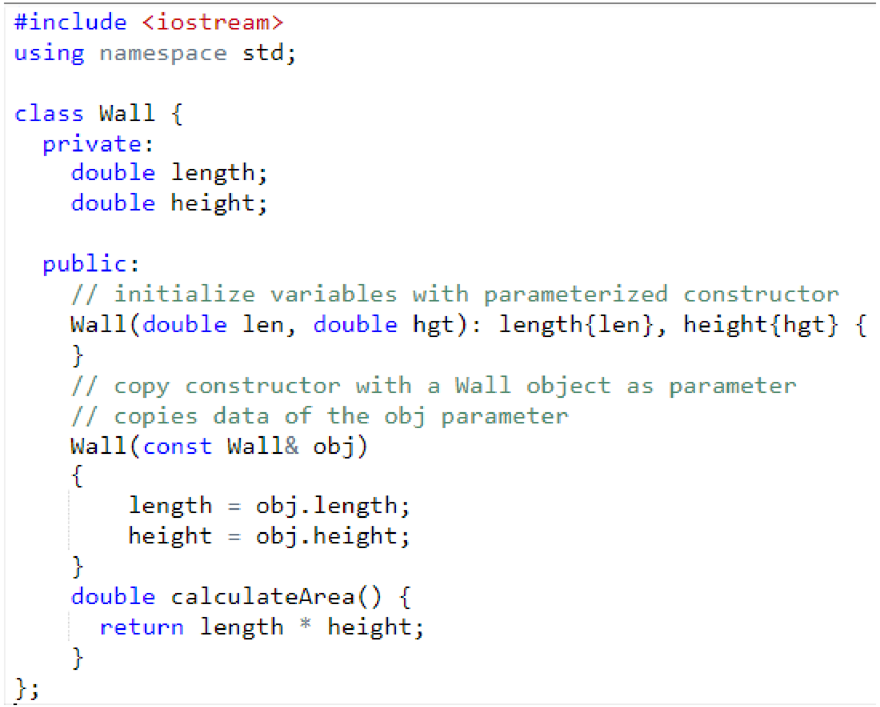


```
int main()
{
    float area;
    circle c1(14.4);
    area = c1.calculateArea();
    cout << "Area: "<< area;
    return 0;
}
```

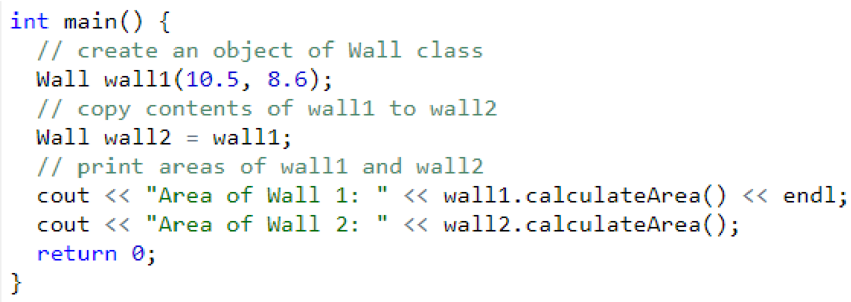
Copy Constructor

- A C++ copy constructor creates an object by initializing it with an existing object of the same class.
- Simply, it creates a new object with the same values as an existing object.

Code Example

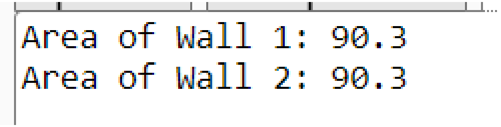


```cpp
#include <iostream>
using namespace std;

class Wall {
  private:
    double length;
    double height;

  public:
    // initialize variables with parameterized constructor
    Wall(double len, double hgt): length{len}, height{hgt} {
    }
    // copy constructor with a Wall object as parameter
    // copies data of the obj parameter
    Wall(const Wall& obj)
    {
        length = obj.length;
        height = obj.height;
    }
    double calculateArea() {
      return length * height;
    }
};
```



```cpp
int main() {
  // create an object of Wall class
  Wall wall1(10.5, 8.6);
  // copy contents of wall1 to wall2
  Wall wall2 = wall1;
  // print areas of wall1 and wall2
  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();
  return 0;
}
```

- **Output**

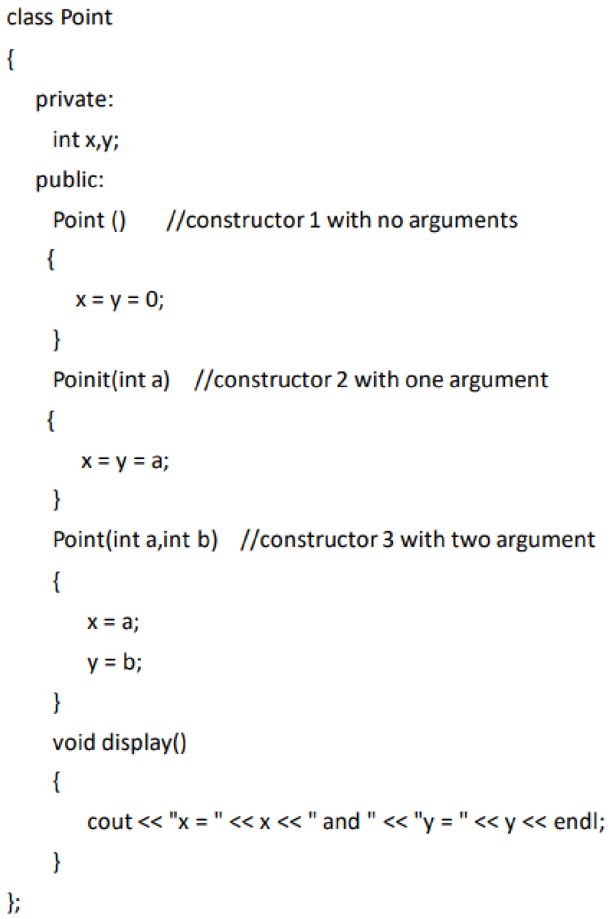


```
Area of Wall 1: 90.3
Area of Wall 2: 90.3
```

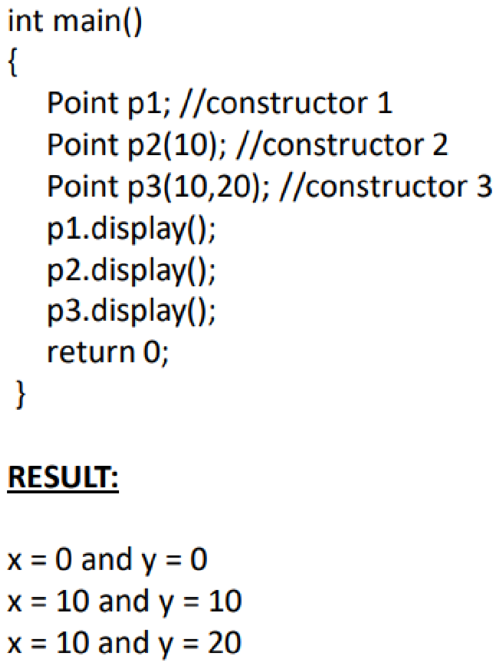
Constructor Overloading

- Using multiple constructors having same name(class name) with different arguments is called constructor overloading.
- Depending upon the number and type of arguments passed, the corresponding constructor is called.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called

Code Example



```
class Point
{
    private:
      int x,y;
    public:
      Point ()      //constructor 1 with no arguments
     {
         x = y = 0;
      }
      Poinit(int a)    //constructor 2 with one argument
     {
          x = y = a;
      }
      Point(int a,int b)    //constructor 3 with two argument
      {
           x = a;
           y = b;
      }
      void display()
      {
           cout << "x = " << x << " and " << "y = " << y << endl;
      }
};
```



```
int main()
{
    Point p1; //constructor 1
    Point p2(10); //constructor 2
    Point p3(10,20); //constructor 3
    p1.display();
    p2.display();
    p3.display();
    return 0;
 }
```

RESULT:

```
x = 0 and y = 0
x = 10 and y = 10
x = 10 and y = 20
```

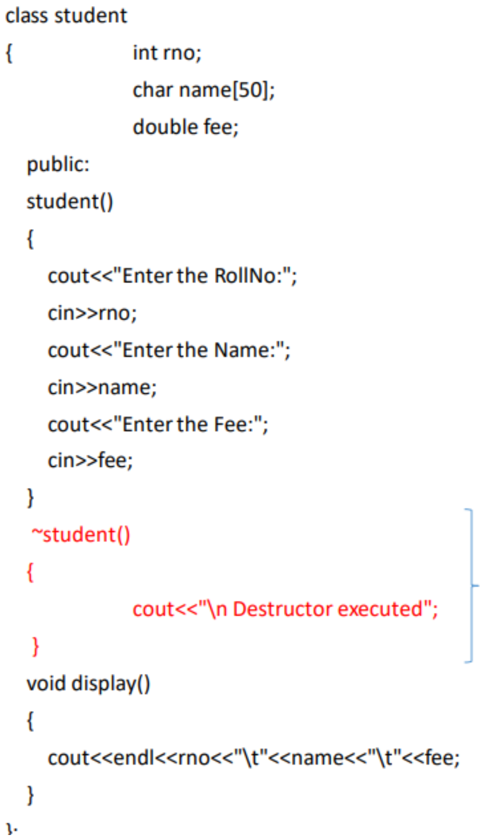
Destructor

- A destructor is a member function that is automatically called when an object is destroyed.
- It is called implicitly when an object is destroyed.
- A class's destructor is a special method that is automatically called when an object of that class is destroyed or goes out of scope.
- Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.
- Destructors have the same name as the class, preceded by a tilde character (~).
- Destructor neither requires any argument nor returns any value.

Destructor

- They handle shutdown procedures as opposed to constructors setting up during object creation.
- Destructor release memory space occupied by the objects created by the constructor.
- Common use of destructors includes freeing dynamically allocated memory by the class object.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object create by constructor. Hence destructor cannot be overloaded
- **Example:** ~Rectangle is the destructor for the Rectangle class.

Code Example



```
class student
{               int rno;
                char name[50];
                double fee;
  public:
  student()
  {
    cout<<"Enter the RollNo:";
    cin>>rno;
    cout<<"Enter the Name:";
    cin>>name;
    cout<<"Enter the Fee:";
    cin>>fee;
  }
   ~student()
  {
                cout<<"\n Destructor executed";
   }
  void display()
  {
    cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;
  }
};
```

```
int main()
{
   student s;
  s.display();
   return 0;

}
```

USER DEFINED DEFAULT DESTRUCTOR OF CLASS STUDENT