OOP

Lecture 9

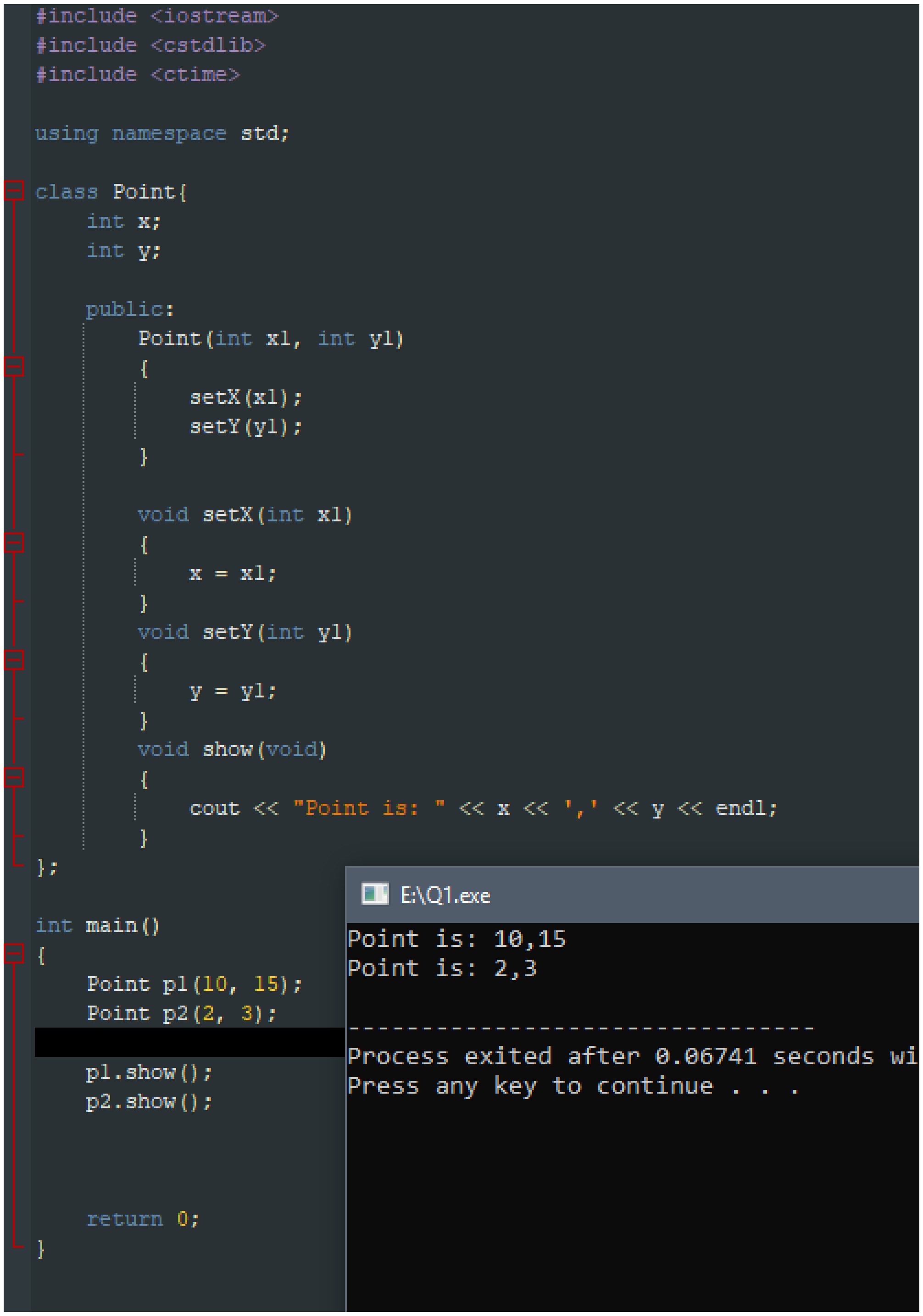
4/11/2021

Last Lecture was ended on constructors. Now we are continuing further,

Constructors:

A constructor in C++ is a special 'MEMBER FUNCTION'

having the same name as that of its class which is used to initialize some valid values to the data members of an object. It is executed automatically whenever an object of a class is created. The only restriction that applies to the constructor is that it must not have a return type or void. It is because the constructor is automatically called by the compiler and it is normally used to INITIALIZE VALUES. The compiler distinguishes the constructor from other member functions of a class by its name which is the same as that of its class. The constructor can be defined as a class in the same way as that of normal member functions and can access any of its data members. The syntax for defining constructor inside the class body is as follows:



```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

class Point{
    int x;
    int y;

    public:
        Point(int x1, int y1)
        {
            setX(x1);
            setY(y1);
        }

        void setX(int x1)
        {
            x = x1;
        }
        void setY(int y1)
        {
            y = y1;
        }
        void show(void)
        {
            cout << "Point is: " << x << ',' << y << endl;
        }
};

int main()
{
    Point p1(10, 15);
    Point p2(2, 3);

    p1.show();
    p2.show();



    return 0;
}
```

```
Point is: 10,15
Point is: 2,3

--------------------------------
Process exited after 0.06741 seconds wi
Press any key to continue . . .
```

Default Constructor :

If we define objects and classes without defining any constructor for a class. In such a situation, the compiler automatically generates a constructor of its own without any parameters known as Default Constructor. The compiler generates a default constructor which is invoked and called automatically whenever any object of the class is created but doesn’t perform any initialization. However, if we define a default constructor explicitly, the compiler no longer generates a default constructor for us.

Types of Constructors :

There are basically two types of constructors

1. **Non - parameterized constructor**
2. **Parameterized constructor**

- A constructor to which no arguments are passed is called the Non-parameterized constructor constructor. It is also called a constructor with no parameters.

- It is possible to pass one or more arguments to a constructor. Constructors that can take arguments are known as parameterized constructors.

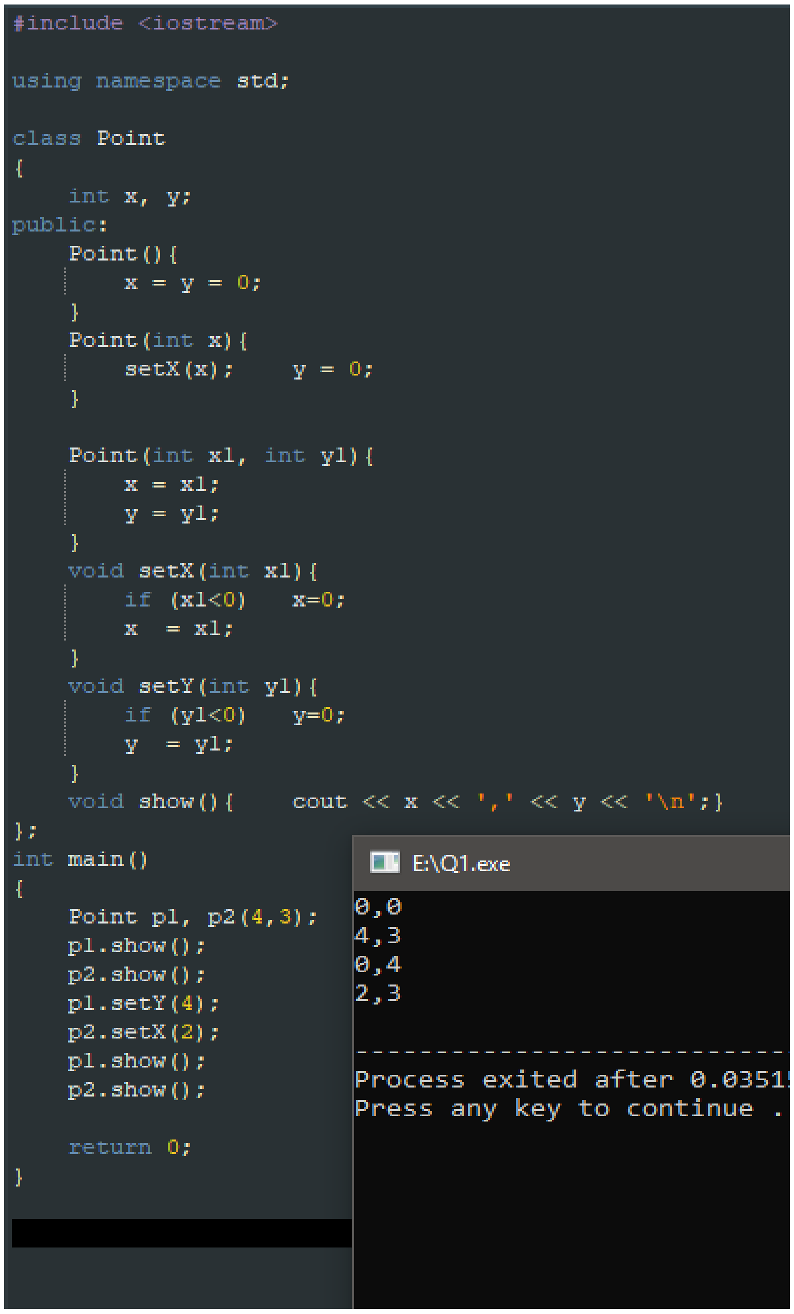
Uses of Constructors :

- Using the constructors, data members can be initialized to some realistic values in its definition even if no arguments are specified explicitly.
- A constructor can preferably be used for initialization and not for input/output operations.
- Constructors are also used to locate memory at run time using the new operator.
- A constructor is executed repeatedly whenever the objects of a class are created.

Constructor Overloading :

Constructor overloading means we can declare more than one constructor in a class. In some programs, a class had only one constructor which had either zero, one, or more parameters. The constructor is key for object initialization. The mechanism of the constructor has been made more powerful by uniting with the feature of overloading. It is made possible by providing more than one constructor in a class called Constructor overloading. We can define as many Constructors as we want by specifying a different number of parameters in each constructor. When we call a constructor, it will be decided automatically which constructor is to call by the number of the arguments that we pass to the constructor.

Sample Program :



```cpp
#include <iostream>

using namespace std;

class Point
{
    int x, y;
public:
    Point(){
        x = y = 0;
    }
    Point(int x){
        setX(x);    y = 0;
    }

    Point(int x1, int y1){
        x = x1;
        y = y1;
    }
    void setX(int x1){
        if (x1<0)    x=0;
        x  = x1;
    }
    void setY(int y1){
        if (y1<0)    y=0;
        y  = y1;
    }
    void show(){    cout << x << ',' << y << '\n';}
};
int main()
{
    Point p1, p2(4,3);
    p1.show();
    p2.show();
    p1.setY(4);
    p2.setX(2);
    p1.show();
    p2.show();

    return 0;
}
```

E:\Q1.exe

```
0,0
4,3
0,4
2,3

----------------------------
Process exited after 0.0351
Press any key to continue .
```

Destructors :

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object. A destructor is automatically called when The program finishes its execution or the function into which the object of the class was created has ended. Similar to constructor, the destructor name should exactly match with the class name. A destructor declaration should always begin with the tilde(~) symbol. Unlike Constructors, There cannot be more than one destructor in a class, destructors do not allow any parameter. When we do not specify any destructor in a class, the compiler generates a default destructor and inserts it into our code.

Syntax :

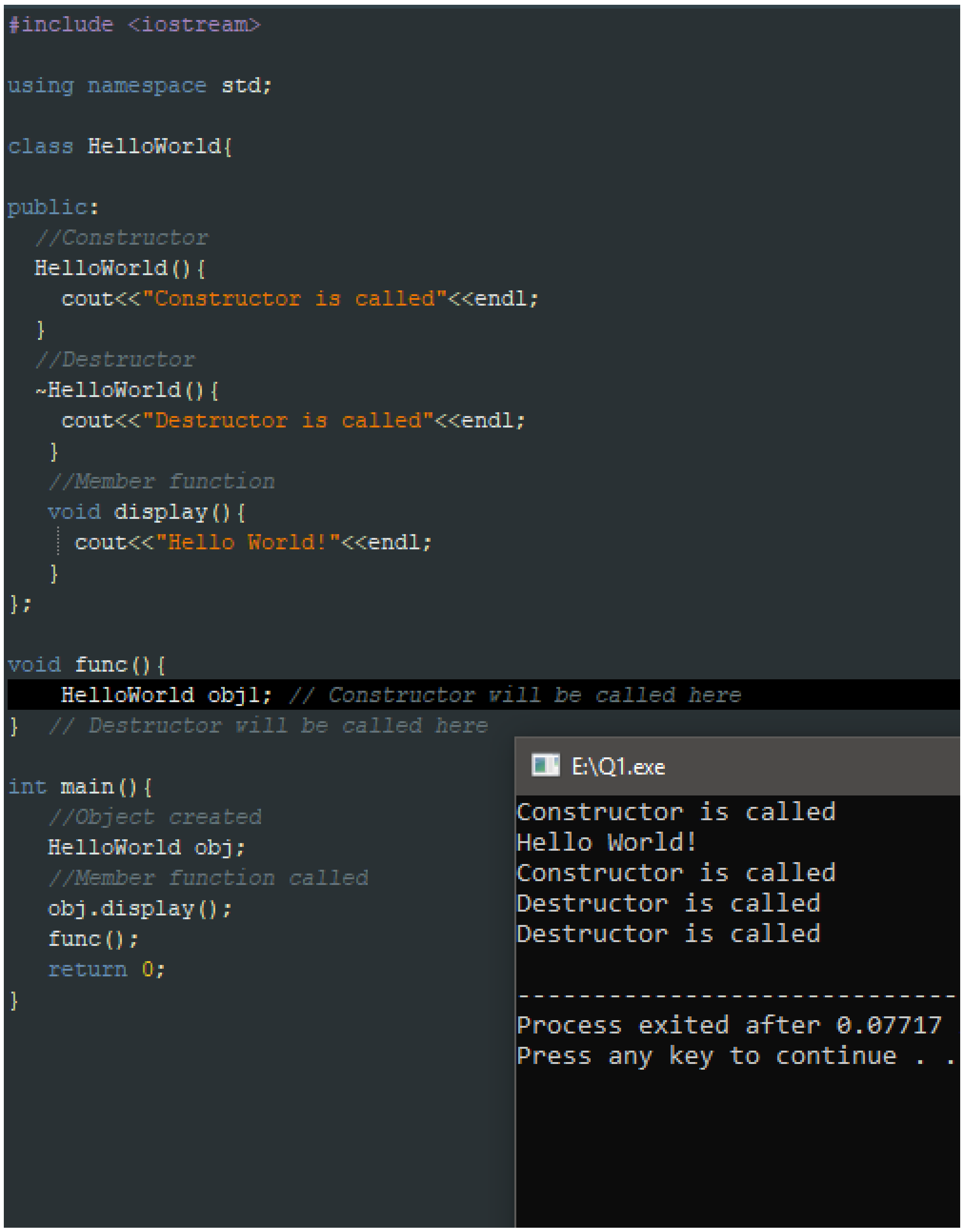
```
~class_name() {

        ------------

        -----------

}
```

Sample Program :



```
#include <iostream>

using namespace std;

class HelloWorld{

public:
  //Constructor
  HelloWorld(){
    cout<<"Constructor is called"<<endl;
  }
  //Destructor
  ~HelloWorld(){
    cout<<"Destructor is called"<<endl;
   }
   //Member function
   void display(){
     cout<<"Hello World!"<<endl;
   }
};

void func(){
    HelloWorld obj1; // Constructor will be called here
}  // Destructor will be called here

int main(){
   //Object created
   HelloWorld obj;
   //Member function called
   obj.display();
   func();
   return 0;
}
```

E:\Q1.exe

```
Constructor is called
Hello World!
Constructor is called
Destructor is called
Destructor is called

--------------------------------
Process exited after 0.07717
Press any key to continue . .
```

Constant Functions :

The const member functions are the functions which are declared as constant in the program. The object called by these functions cannot be modified. In other words, we cannot change or modify the data members of the class in const function. It is recommended to use const keywords so that accidental changes to objects are avoided. A const member function can be called by any type of object. Non-const functions can be called by non-const objects only.

Copy Constructor :

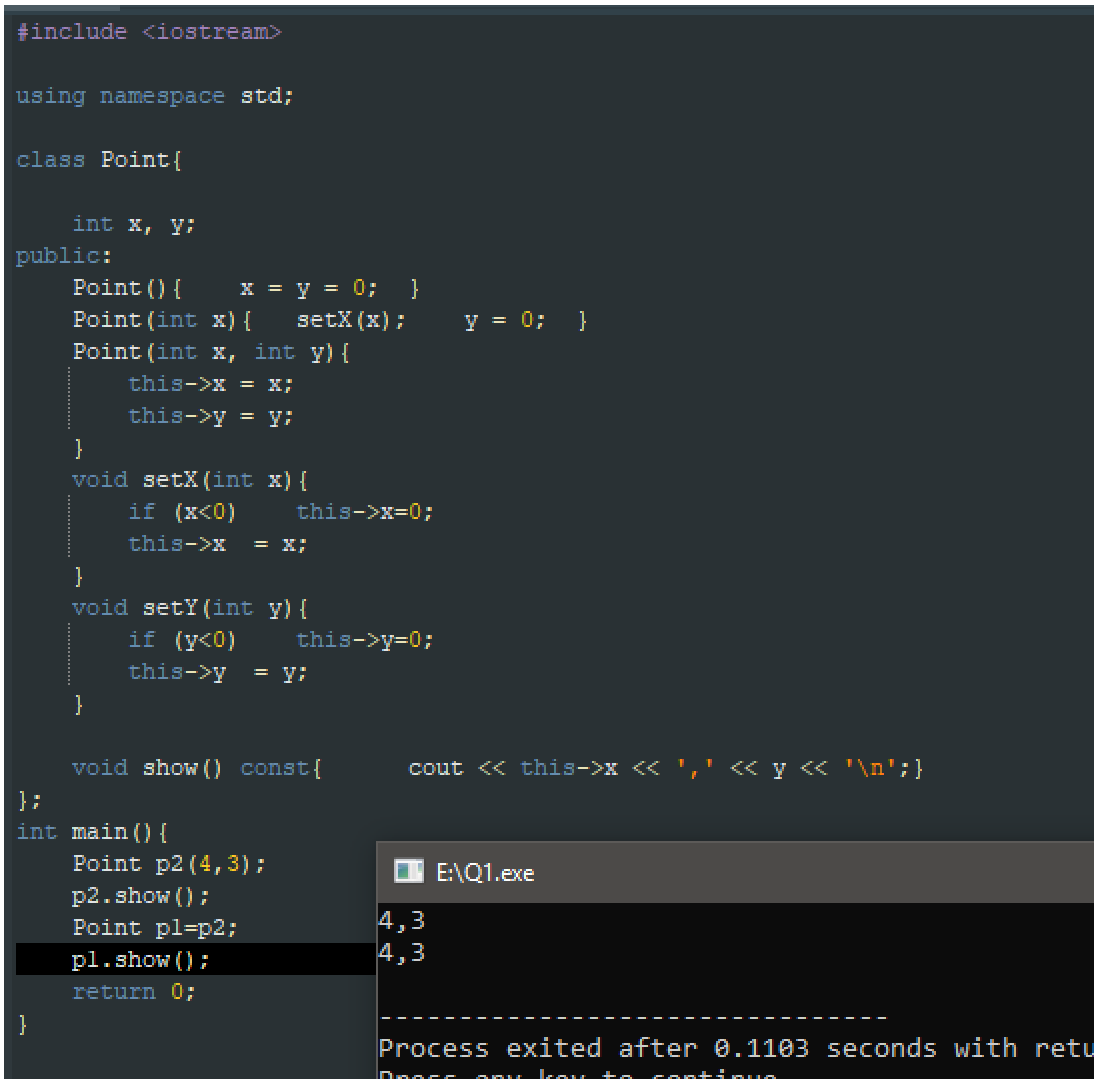
The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to:

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Initialize one object from another of the same type.

If a copy constructor is not defined explicitly in a class, the compiler itself defines one which is known as default copy constructor.

this-> Pointer :

this pointer exists by default in every class and in all members. this Pointer points to the current object with which the member function is called. All the data members of the objects will be accessed by this->variaable_name;



```cpp
#include <iostream>

using namespace std;

class Point{

    int x, y;
public:
    Point(){    x = y = 0;  }
    Point(int x){   setX(x);    y = 0;  }
    Point(int x, int y){
        this->x = x;
        this->y = y;
    }
    void setX(int x){
        if (x<0)    this->x=0;
        this->x  = x;
    }
    void setY(int y){
        if (y<0)    this->y=0;
        this->y  = y;
    }

    void show() const{      cout << this->x << ',' << y << '\n';}
};
int main(){
    Point p2(4,3);
    p2.show();
    Point p1=p2;
    p1.show();
    return 0;
}
```

E:\Q1.exe

```
4,3
4,3

--------------------------------
Process exited after 0.1103 seconds with retu
```

Uses of this Pointer :

- It is used to access data members, if local variables and the data members have the same names.
- It is used to return the current object from Functions.

--THE END--