OOP
Lecture 12

16/11/2021

Quiz 5 :

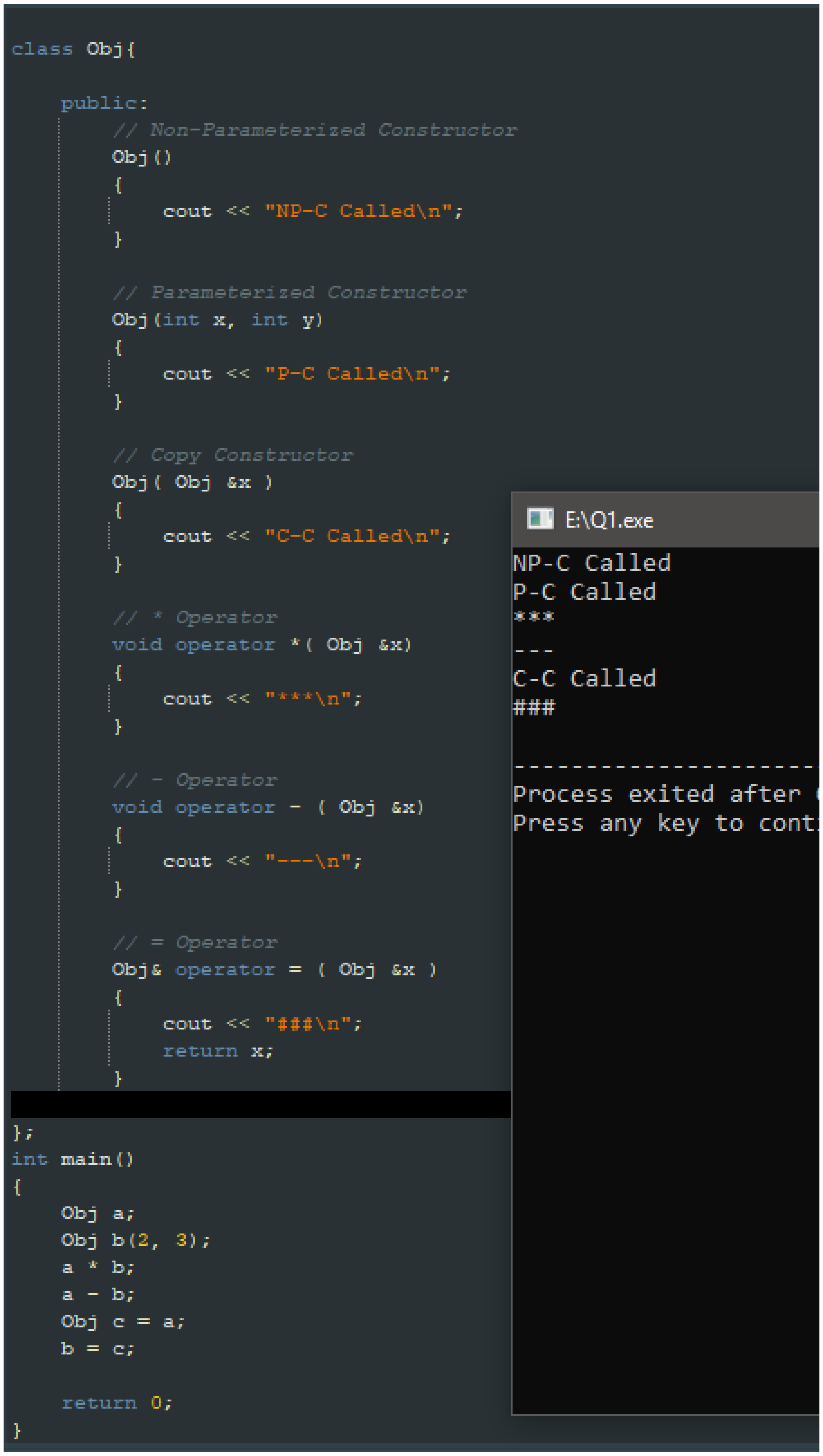
Task 1 : Main() function and Output is given, Write member Functions of the class.

```
int main()
{
        Obj a;                                    |       NP-C Called
        Obj b(2, 3);                              |       P-C Called
        a * b;                                    |       ***
        a - b;                                    |       ---
        Obj c = a;                                |       C-C Called
        b = c;                                    |       ###

        return 0;
}
```

Solution:

```
        public:
                // Non-Parameterized Constructor
                Obj()
                {
                        cout << "NP-C Called\n";
                }
```

```
// Parameterized Constructor
Obj(int x, int y)
{
        cout << "P-C Called\n";
}

// Copy Constructor
Obj( Obj &x )
{
        cout << "C-C Called\n";
}

// * Operator
void operator *( Obj &x)
{
        cout << "***\n";
}

// - Operator
void operator - ( Obj &x)
{
        cout << "---\n";
}

// = Operator
Obj& operator = ( Obj &x )
{
        cout << "###\n";
        return x;
}
```



```
class Obj{

    public:
        // Non-Parameterized Constructor
        Obj()
        {
            cout << "NP-C Called\n";
        }

        // Parameterized Constructor
        Obj(int x, int y)
        {
            cout << "P-C Called\n";
        }

        // Copy Constructor
        Obj( Obj &x )
        {
            cout << "C-C Called\n";
        }

        // * Operator
        void operator *( Obj &x)
        {
            cout << "***\n";
        }

        // - Operator
        void operator - ( Obj &x)
        {
            cout << "---\n";
        }

        // = Operator
        Obj& operator = ( Obj &x )
        {
            cout << "###\n";
            return x;
        }
[redacted]
};
int main()
{
    Obj a;
    Obj b(2, 3);
    a * b;
    a - b;
    Obj c = a;
    b = c;

    return 0;
}
```

E:\Q1.exe

```
NP-C Called
P-C Called
***
---
C-C Called
###

----------------------
Process exited after
Press any key to cont
```

Task 2: Write a class Matrix3By3 with one data member which is a static 2d array of 3 by 3. Write Following member functions:

1. Check if the passed row number is valid, return the sum of that row, otherwise return minimum integer value.
2. Check if the passed row number is valid, then check if all the numbers in a row are 1, then return 1, otherwise return 0, if row number is invalid return minimum integer value.
3. Pass 2 rows i, j to function. Assume that they are valid. Swap these two rows.
4. Rotate the columns of the matrix such that 2nd column becomes first, 3rd column becomes 2nd and 1st column becomes 3rd.

Solution:

```
class Matrix3By3{

        int m[3][3];

        public:
                int sumOfRow(int i)
                {
                        int j, sum = 0;
                        if ( i < 0 || i > 2 )
                                return INT_MIN;

                        for ( j = 0; j < 3; j++ )
                                sum += m[i][j];

                        return sum;
                }
```

```
int isAllOne( int i )
{
        int j;


        if ( i < 0 || i > 2 )
                return INT_MIN;

        for ( j = 0; j < 3; j++ )
                if ( m[i][j] != 1 )
                        return 0;

        return 1;
}

Matrix3By3& swapRow( int i, int j )
{
        int temp, k;

        for ( k = 0; k < 3; k++ )
        {
                temp = m[i][k];
                m[i][k] = m[j][k];
                m[j][k] = temp;
        }

        return *this;
}

Matrix3By3& rotateCol( )
{
        int temp[3], t, i, j;
```

```
                        for ( i = 0; i < 3; i++)
                                temp[i] = m[i][0];

                        for ( i = 1; i < 3; i++ )
                                for( j = 0; j < 3; j++ )
                                        {
                                                t = m[j][i];
                                                m[j][i] = m[j][i-1];
                                                m[j][i-1] = t;
                                        }

                        for ( i = 0; i < 3; i++)
                                m[i][2] = temp[i];

                        return *this;
                }
};
```

QUIZ ENDED

Lecture 12

Array of Objects:

Like an array of other data types(like int, float), an array of type class (User-Defined data type) can also be created. The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type.

Syntax :

```
        class_name   array_name [size] ;
```

The array of objects is beneficial when we need to create many objects. By using an array of objects, We can create them with the same name and use them in a single loop. When an array of objects is created, it calls the default constructor or the non-parameterized constructor from the class. The constructor is called once for every object so the total number of times the constructor is called is the same as the size of the array of objects. Like, if an array of 5 objects is created, it calls the default or non-parameterized constructor 5 times. If we are not defining any constructor in the class, then it will call the default constructor, and if we have defined any constructor, then we have to define a non-parameterized constructor explicitly. If only a parameterized constructor exists in a class, i.e no non-parameterized constructor then it will give an error.

We can call also parameterized constructor for some specific objects in an array. Like this,

Suppose we have a Point class defined and we are creating an array of 5 objects of class Point. Now,

Point arr[5];   // Calls default or non-parameterized constructor 5 times

Point arr[5] = {Point(2, 3), Point(), Point(1), Point(3, 4)};

// Calls the 2 parametric constructor for the first object i.e at index 0(arr[0]).

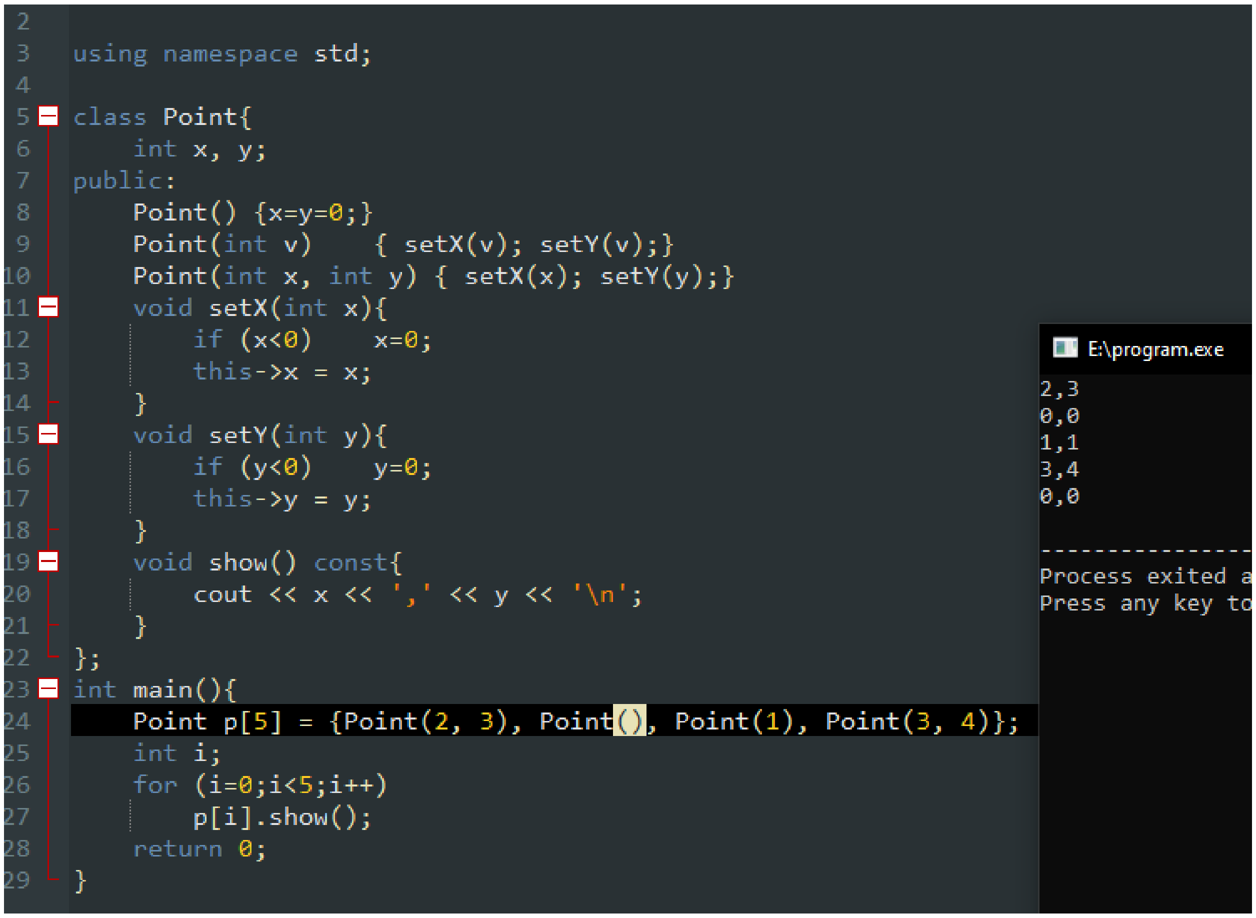
Then Calls default or non-parameterized constructor for the second object i.e at index 1 (arr[1]).

Then Calls the 1 parametric constructor for the third object i.e at index 2 (arr[2]).

Then Calls the 2 parametric constructor for the forth object i.e at index 3(arr[3]).

And then the default or non-parameterized constructor will be called for the remaining objects, in this case only the fifth object is left at index 4 (arr[4]).

Sample Program :



```cpp

using namespace std;

class Point{
    int x, y;
public:
    Point() {x=y=0;}
    Point(int v)    { setX(v); setY(v);}
    Point(int x, int y) { setX(x); setY(y);}
    void setX(int x){
        if (x<0)    x=0;
        this->x = x;
    }
    void setY(int y){
        if (y<0)    y=0;
        this->y = y;
    }
    void show() const{
        cout << x << ',' << y << '\n';
    }
};
int main(){
    Point p[5] = {Point(2, 3), Point(), Point(1), Point(3, 4)};
    int i;
    for (i=0;i<5;i++)
        p[i].show();
    return 0;
}
```

E:\program.exe

```
2,3
0,0
1,1
3,4
0,0

----------------
Process exited a
Press any key to
```

Static Class Member / Static Data member of class :
We can declare any data member(Variable) of the class as static by using the **static** keyword before data type.

Like, **static int count;**

When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member. A static member is shared by all objects of the class. We can't initialize a static data member in the class definition(Class Brackets) but it can be initialized outside the class using the scope resolution operator :: to identify which class it belongs to. We can use the static variable directly with the class name without any object. We can also access the static member from the main() function if it is public.

Syntax:

```
class name_of_class{

        static int name_of_variable;

        ---------------------
        ---------------------
};

int name_of_class :: name_of_variable = value;
```

Example :

```
class Point{

        static int count;

        ---------------------
        ---------------------
};

int Point :: count = 0;
```

Use :

The most common use of static variables is a count. Such variables store the count of the total objects that are created. To count to total objects we can increment our count variables in constructors (including parametric, non-parametric, copy constructor) and decrement the count in destructors.

Static Function :

Like static member variables, we can also make functions static. We can declare any member function of the class as static by using the static keyword before data type.

Like, **static int getCount()** { -------------------- }

- We can access the static function directly by using the class name and scope resolution operator.
- Inside a static function, we can not use any non-static data member of the class. Because non-static members must belong to a class object, and static member functions have no class object to work with.
- Static member functions are not attached to an object, it means they have no “this” pointer. This makes sense when you think about it, “this” pointer always points to the object that the member function is working on. Static member functions do not work on an object (i.e static functions can not use the data members of the other objects which are always non-static), so, “this” pointer is not needed.

Note :

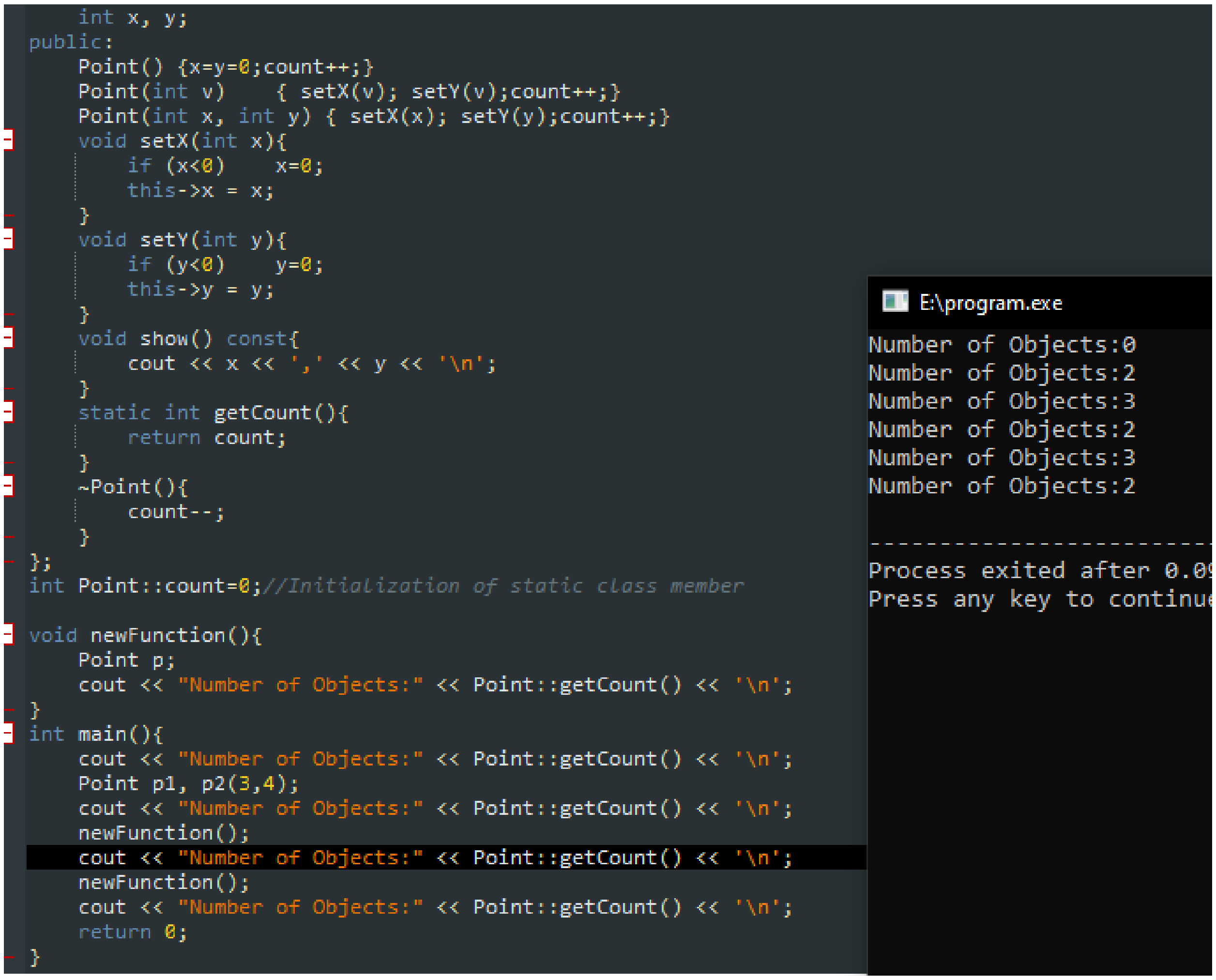
Here's a specific Concept if you know the scope and lifetime of the local and global variables. “**Static variables have lifetime of global variables and scope of local variables**”, if you don’t know about these terms, don’t be more confused just skip this point. 😉

An object only exists until the function in which it was created exists. When a function ends all the objects that are created inside that function are destroyed and the destructor of all of those objects is called.

When we call a new function it does not end the previous function, a function only ends when its closing bracket i.e (}) is executed, or any return statement in the function is executed.

In the example below, when we called the newFunction and created an object “p” , when we printed number of objects inside the function, it was 3 and when we returned to main function it was again 2, because when the newFunction ended it called the destructor of the object p and decrement the total count (as the object “p” is also destroyed with the function).
The Objects created in the main function are only destroyed when the “**return 0;**” statement of the function is executed (it is exactly that point when the program is also terminated).

Sample Program to count Number of Existing objects :



```cpp
    int x, y;
public:
    Point() {x=y=0;count++;}
    Point(int v)    { setX(v); setY(v);count++;}
    Point(int x, int y) { setX(x); setY(y);count++;}
    void setX(int x){
        if (x<0)    x=0;
        this->x = x;
    }
    void setY(int y){
        if (y<0)    y=0;
        this->y = y;
    }
    void show() const{
        cout << x << ',' << y << '\n';
    }
    static int getCount(){
        return count;
    }
    ~Point(){
        count--;
    }
};
int Point::count=0;//Initialization of static class member

void newFunction(){
    Point p;
    cout << "Number of Objects:" << Point::getCount() << '\n';
}
int main(){
    cout << "Number of Objects:" << Point::getCount() << '\n';
    Point p1, p2(3,4);
    cout << "Number of Objects:" << Point::getCount() << '\n';
    newFunction();
    cout << "Number of Objects:" << Point::getCount() << '\n';
    newFunction();
    cout << "Number of Objects:" << Point::getCount() << '\n';
    return 0;
}
```

E:\program.exe

```
Number of Objects:0
Number of Objects:2
Number of Objects:3
Number of Objects:2
Number of Objects:3
Number of Objects:2

-------------------------
Process exited after 0.0
Press any key to continu
```

Dynamic Objects of the Class :

We can also create the dynamic objects of the class, i.e first we need to create a pointer to the class object and then we can use the **new** operator to define a dynamic object.

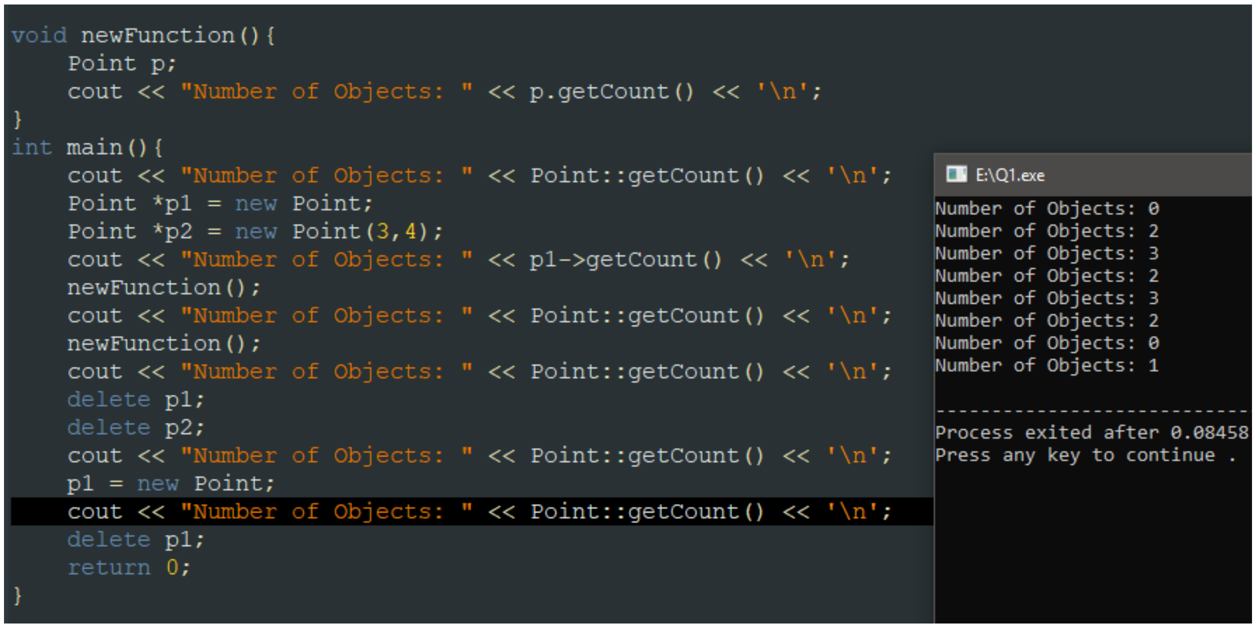
Syntax :

class_name *pointer_name;
pointer_name = new class_name;

OR. pointer_name = new class_name(parameters); // to call parameterized constructor

- The key benefit of using dynamic objects is that we can create new objects at runtime and destroy the object when we no longer need it.
- delete operator is used to delete the dynamic objects.
- The destructor of the object is called when an object is deleted.
- The pointer to the object will not be deleted, it means that we can create a new object in the same pointer again.
- We can also create dynamic arrays by combining the concepts of array of objects and dynamic objects.
- To call the member function of the class we use the (->) operator instead of the dot (**.**) operator.

Sample Program :



```
void newFunction(){
    Point p;
    cout << "Number of Objects: " << p.getCount() << '\n';
}
int main(){
    cout << "Number of Objects: " << Point::getCount() << '\n';
    Point *p1 = new Point;
    Point *p2 = new Point(3,4);
    cout << "Number of Objects: " << p1->getCount() << '\n';
    newFunction();
    cout << "Number of Objects: " << Point::getCount() << '\n';
    newFunction();
    cout << "Number of Objects: " << Point::getCount() << '\n';
    delete p1;
    delete p2;
    cout << "Number of Objects: " << Point::getCount() << '\n';
    p1 = new Point;
    cout << "Number of Objects: " << Point::getCount() << '\n';
    delete p1;
    return 0;
}
```

E:\Q1.exe

```
Number of Objects: 0
Number of Objects: 2
Number of Objects: 3
Number of Objects: 2
Number of Objects: 3
Number of Objects: 2
Number of Objects: 0
Number of Objects: 1

--------------------------------
Process exited after 0.08458
Press any key to continue .
```

Unary Operators :

Unary operators are the operators that operate on a single operand to give the specific result. To perform the operations on these operators we need to use only a single operand. However, the operands which use two operands are known as binary operators.

Example : ++, --, ! are examples of unary operators.

++ Operator :

In programming, the increment operator ++ increases the value of a variable by 1.

It means that if we write, p++; we are trying to do p = p + 1;

If we try to use the ++ operator with the object of a class, Suppose we are again going with our Point Class (unfortunately 😓), what will happen when we do p++; if p is an object of our Point class?

Remember? We studied Operator Overloading once? 👀

The compiler will give a full fledged **ERROR**…. Saying what can i do with this operator…. So, the solution is we need to overload the ++ operator first, in order to use it with the object of our class.

- The first point which comes to mind is that if it is a unary operator (i.e if it works only with one operand), it does not need any parameter, it can manipulate and use the current/caller object.

- The logic to implement the increment operator changes with class to class. It means that if we are working with a Point class then we need to increment both of its coordinates(x, y) by 1. But, what if we are working with the Rational Number class? In such a case we need to add 1 to our whole Rational Number. Then, the LCM and all that begins to add 1 to the object. But, it is a story of another lecture, So, typically we will say, "We will study this later 😅", However, Deep down We all know this later will never come in our career😏. But let's keep it our secret. 😎

Anyhow, We can use the same concept to to increment the object of our class, i.e by incrementing both of the x and y coordinates of the object.

Code of ++ Operator :

```
Point& operator ++ ()
        {
                x++;
                y++;
                return *this;
        }
```

If we remember our PF concepts, we know that there are 2 increment operators, i.e Prefix and Postfix increment operators.

Postfix increment operator means the expression is evaluated first using the original value of the variable and then the variable is incremented(increased).

Prefix increment operator means the variable is incremented first and then the expression is evaluated using the new value of the variable.

Note : Just a quick note, but i know everybody has an idea that if we are incrementing a variable in 1 statement i.e we are just doing

```
x++;   OR.   ++x;
```

in a statement and no other expression than there is no difference in postfix and prefix because the statement is completed, the variable will be incremented and we didn't use the previous value of the variable.

The above increment code is the prefix increment code, The code for the postfix incrementation is as follow,

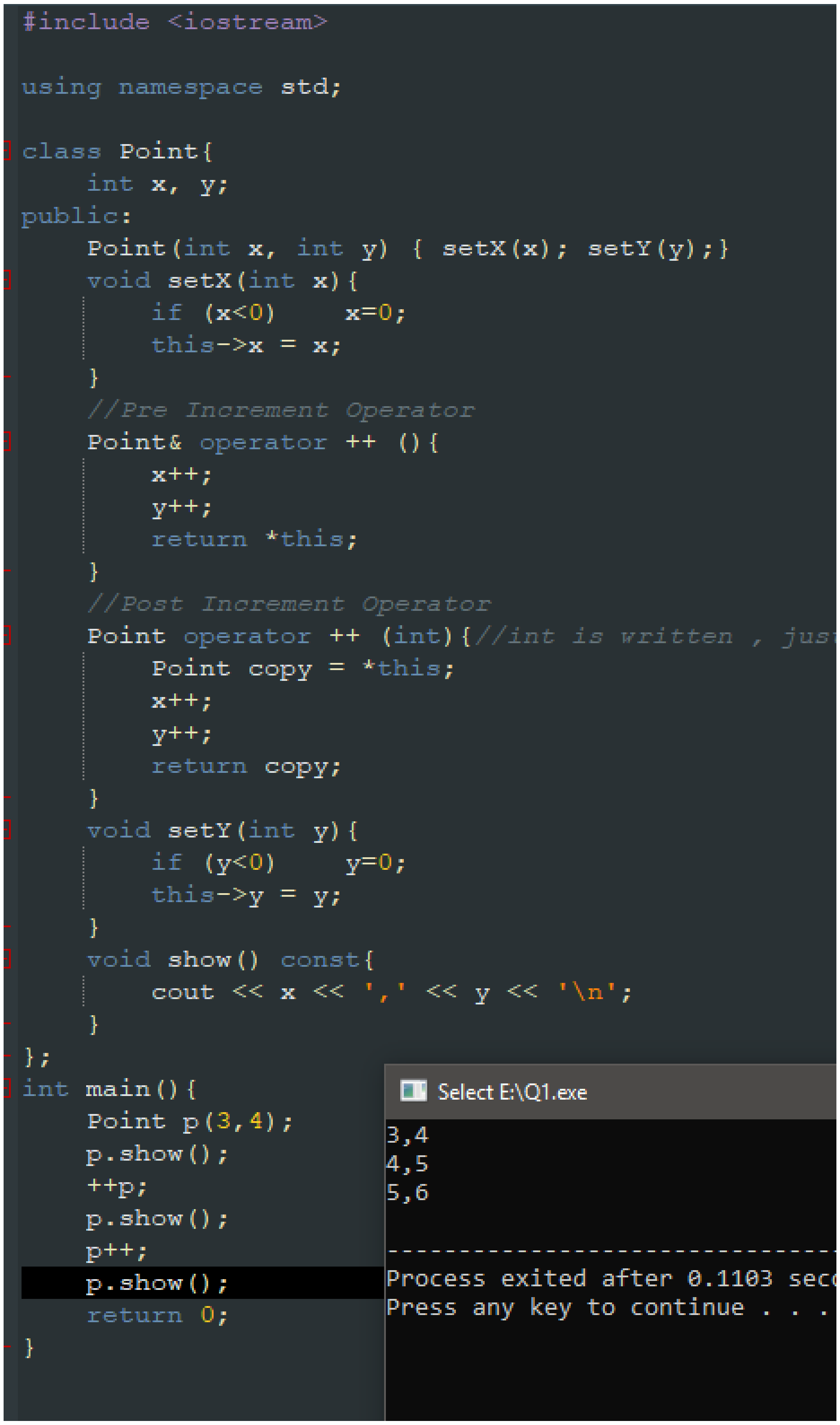
```
Point operator ++ ( int )
        {
                Point copy = *this;
                x++;
                y++;
                return copy;
        }
```

Why did we write “int” in the parenthesis?

int is written inside parenthesis just to differentiate the postfix and prefix increment operators. Maybe, It is not meaningful for us but trust me it is meaningful for the Compiler. It is just a thing to remember that we write “int” inside to parenthesis to tell the compiler that it is a postfix increment operator.

- We can write the Decrement operator with the same concept and syntax. Just change the logic that where we are increasing the value by 1, we will decrease the values by 1.

Sample Program :

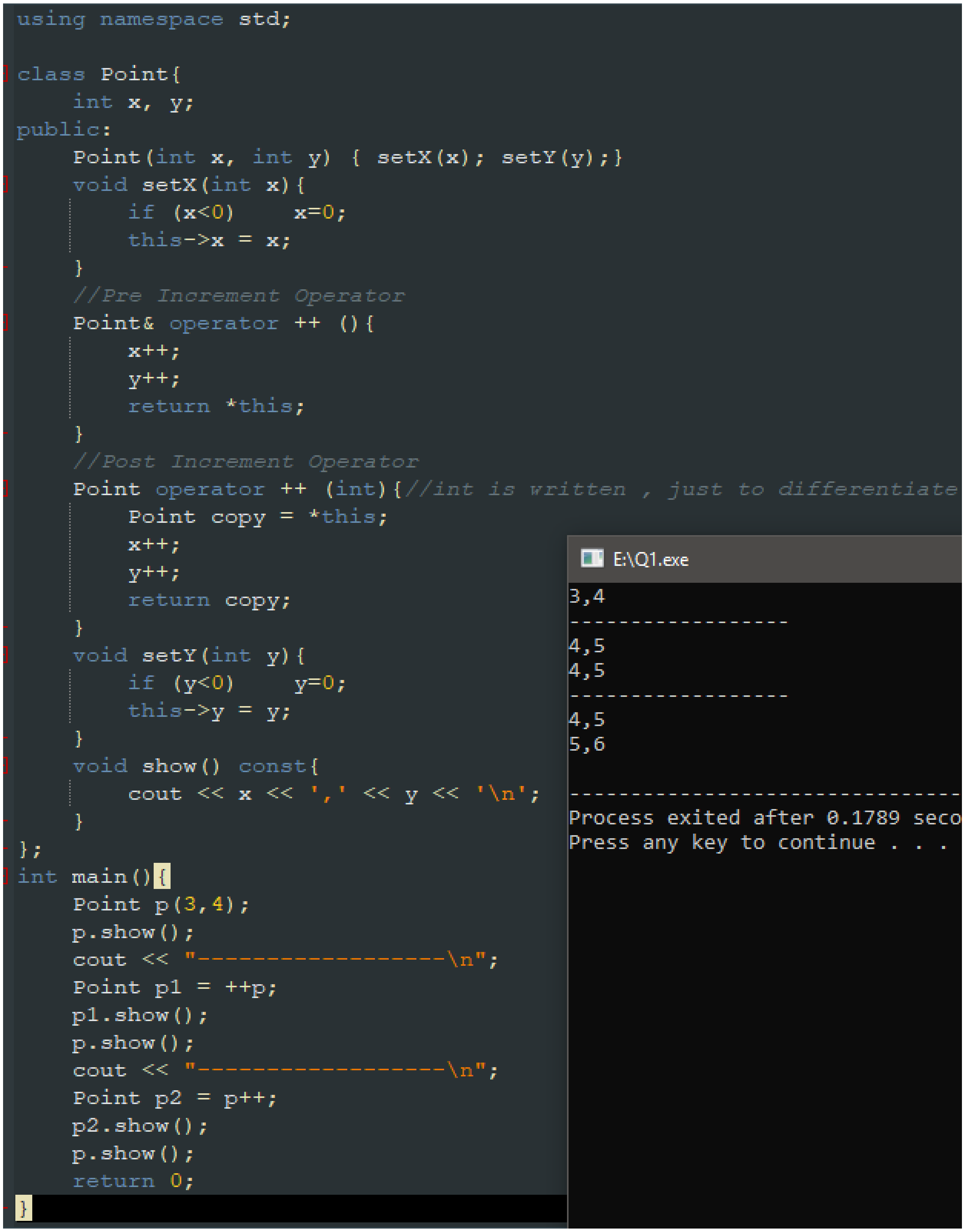


```cpp
#include <iostream>

using namespace std;

class Point{
    int x, y;
public:
    Point(int x, int y) { setX(x); setY(y);}
    void setX(int x){
        if (x<0)    x=0;
        this->x = x;
    }
    //Pre Increment Operator
    Point& operator ++ (){
        x++;
        y++;
        return *this;
    }
    //Post Increment Operator
    Point operator ++ (int){//int is written , jus
        Point copy = *this;
        x++;
        y++;
        return copy;
    }
    void setY(int y){
        if (y<0)    y=0;
        this->y = y;
    }
    void show() const{
        cout << x << ',' << y << '\n';
    }
};
int main(){
    Point p(3,4);
    p.show();
    ++p;
    p.show();
    p++;
    p.show();
    return 0;
}
```

Select E:\Q1.exe

```
3,4
4,5
5,6

--------------------------------
Process exited after 0.1103 sec
Press any key to continue . . .
```

Analyzing Postfix and Prefix incrementing :



```cpp
using namespace std;

class Point{
    int x, y;
public:
    Point(int x, int y) { setX(x); setY(y);}
    void setX(int x){
        if (x<0)    x=0;
        this->x = x;
    }
    //Pre Increment Operator
    Point& operator ++ (){
        x++;
        y++;
        return *this;
    }
    //Post Increment Operator
    Point operator ++ (int){//int is written , just to differentiate
        Point copy = *this;
        x++;
        y++;
        return copy;
    }
    void setY(int y){
        if (y<0)    y=0;
        this->y = y;
    }
    void show() const{
        cout << x << ',' << y << '\n';
    }
};
int main(){
    Point p(3,4);
    p.show();
    cout << "------------------\n";
    Point p1 = ++p;
    p1.show();
    p.show();
    cout << "------------------\n";
    Point p2 = p++;
    p2.show();
    p.show();
    return 0;
}
```

E:\Q1.exe

```
3,4
------------------
4,5
4,5
------------------
4,5
5,6

--------------------------------
Process exited after 0.1789 seco
Press any key to continue . . .
```

Relational Operators :

A relational operator is used to check the relationship between two operands. Relational operators are binary meaning they require two operands. The following table summarizes the relational operators used in C++.

Operator	Meaning	Example
`==`	**Is Equal To**	`3 == 5` **gives us false**
`!=`	**Not Equal To**	`3 != 5` **gives us true**
`>`	**Greater Than**	`3 > 5` **gives us false**
`<`	**Less Than**	`3 < 5` **gives us true**
`>=`	**Greater Than or Equal To**	`3 >= 5` **give us false**
`<=`	**Less Than or Equal To**	`3 <= 5` **gives us true**

The Relational operators return only True or False, Do you remember any data type which comprises only these two states? 😋

There is a data type named “bool” in C++, which has only two values i.e True or False. Mathematically in C++, False means 0 and True means any value other than 0 (both positive and negative) but most probably it is 1.

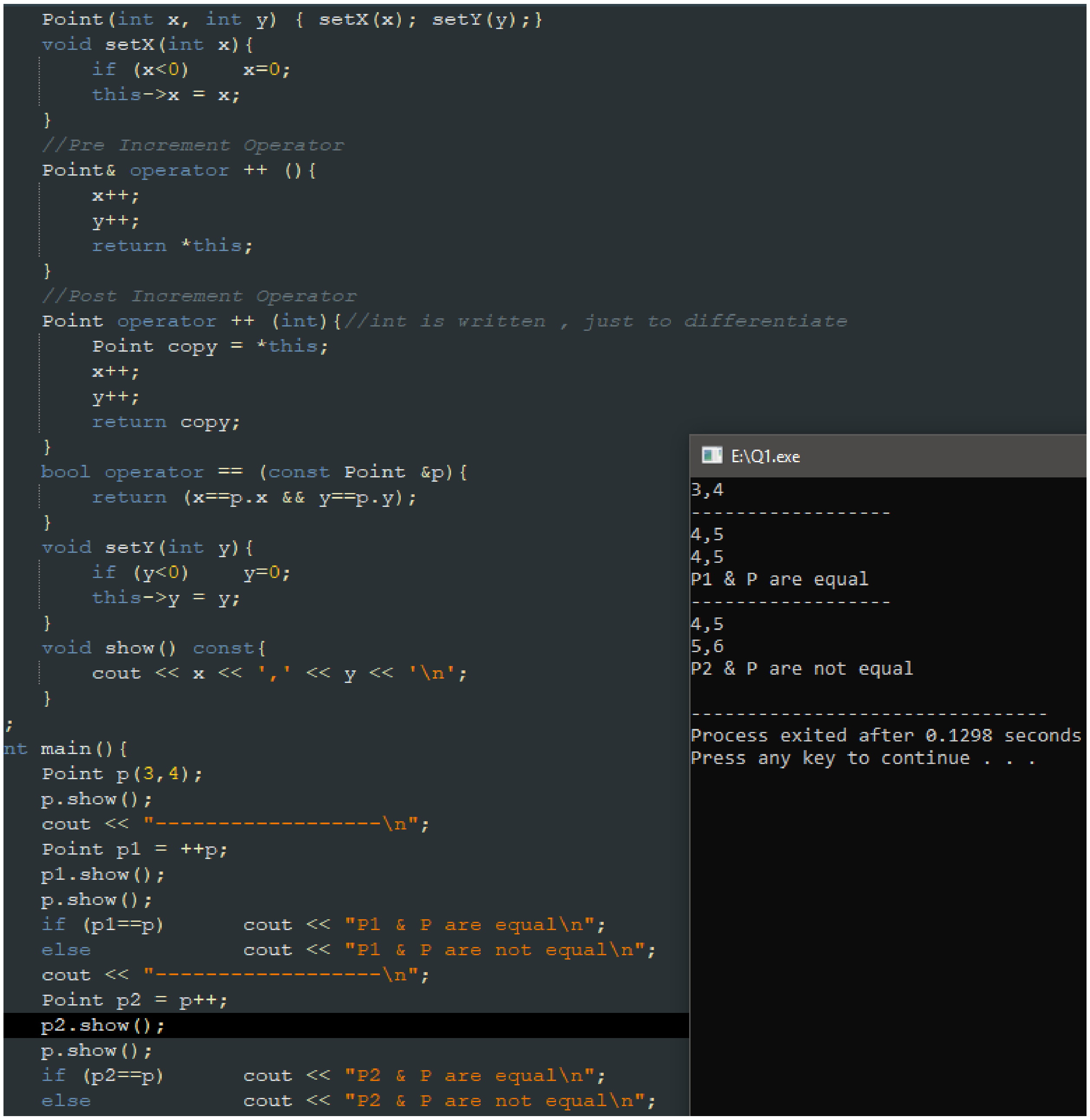
== Operator :

== operator is used to compare two objects, and return if True(1) if both of the objects are equal. The same concept applies here as well. We need to check all the data members of the object, if all of them are the same then return True (1) otherwise False (0).

Code for == Operator :

```
        bool operator == (const Point &p)
{
        return ( x== p.x && y == p.y);
}
```

Sample Program:



```
    Point(int x, int y) { setX(x); setY(y);}
    void setX(int x){
        if (x<0)    x=0;
        this->x = x;
    }
    //Pre Increment Operator
    Point& operator ++ (){
        x++;
        y++;
        return *this;
    }
    //Post Increment Operator
    Point operator ++ (int){//int is written , just to differentiate
        Point copy = *this;
        x++;
        y++;
        return copy;
    }
    bool operator == (const Point &p){
        return (x==p.x && y==p.y);
    }
    void setY(int y){
        if (y<0)    y=0;
        this->y = y;
    }
    void show() const{
        cout << x << ',' << y << '\n';
    }
;
nt main(){
    Point p(3,4);
    p.show();
    cout << "------------------\n";
    Point p1 = ++p;
    p1.show();
    p.show();
    if (p1==p)        cout << "P1 & P are equal\n";
    else              cout << "P1 & P are not equal\n";
    cout << "------------------\n";
    Point p2 = p++;
    p2.show();
    p.show();
    if (p2==p)        cout << "P2 & P are equal\n";
    else              cout << "P2 & P are not equal\n";
```

E:\Q1.exe

```
3,4
------------------
4,5
4,5
P1 & P are equal
------------------
4,5
5,6
P2 & P are not equal

--------------------------------
Process exited after 0.1298 seconds
Press any key to continue . . .
```

--THE END--