

# FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

## Lecture09

### Lecture 9: Constructors in C++ - Copy Constructor

So, welcome to lecture number 9, Classes and Objects. So, in the last lecture, we saw the default constructor and the parameterized constructor, right? So, with the help of the parameterized constructor, can we solve this particular problem, right? So, the problem is to create a complex number class, right? The name of the class is Complex Numbers, and add two complex numbers, right?


It is nothing. So, C1 is A1 plus IB1, right? C2 is equal to A2 plus IB2. In fact, we saw the multiplication. So, C1 plus C2.

So, let us say C3 is C1 plus C2. So, which will be A1 plus A2 plus I into B1 plus B2, right? So, this we know. So, let us solve this using the parameterized constructor, all right. So, the main idea is

So when you are adding two complex numbers, you take one number from the user. So let us say C1 I am taking from the user. And C2 should be passed using the constructor. So you are passing. So when I am passing, it means I have to use it as an argument, right?

**Problem using Parameterized Constructor**

- Create a complex number class and add two complex numbers by taking one number from user and another one should be passed using constructor in C++.

$$\begin{aligned} C_1 &= a_1 + ib_1 \\ C_2 &= a_2 + ib_2 \\ \hline C_3 &= C_1 + C_2 = (a_1 + a_2) + i(b_1 + b_2) \end{aligned}$$


So that means it has the parameter. So that means it is a parameterized constructor. So another input should be passed using the constructor. So let us see how we are going to write this in C++. So let us have the class Complex.

So the class Complex has two member data, real and imaginary. The almost similar thing we have seen, the multiplication of two complex numbers. And here you have the parameterized constructor. So we have seen the name of the class is Complex, and the name of the special member function is also Complex. So, obviously, this is a constructor, and we are using two parameters R and I, the real and imaginary.

So, therefore, you call this a parametric constructor. So, you have used the parametric constructor. So, R will be initialized, right? Whatever you are passing will be initialized to real, and whatever you are passing for I will be initialized to imaginary, correct? So, this is a special member function, a constructor, from line number 11 to 14.

Now, here you go, you have line number 17, correct? So, on line number 17, you have a member function called add, right? You have a member function called add. So, we have two arguments. So, one is C1 and another one is C2.

So, two complex numbers. We have already calculated how to add, that means C1 dot real plus C2 dot real, A1 plus A2, right. So, this is what we worked out: A1 plus A2, that means the real numbers, the real part will be added, and then the imaginary part should be added. So, the real part will be added, put some real, and another variable some imaginary. So, C1 dot imaginary plus C2 dot imaginary, put it in some imaginary.

And you are returning a complex. Some real and some imaginary. Alright. Return the object. Alright.

The name of the object is. The complex with some real. And some imaginary. Correct. So we will see how this is working.


When we are going to talk about the main. So now line number 24. You have white display. One of the member functions. So it is printing real plus.

I into imaginary or imaginary I. Okay. So 2 plus 3 I. That is the usual way. Alright. So that will be displayed over here. So now, what was the problem?

One input you have to take from the user. Alright. So it is asking user real and user imaginary. Alright. So we have a double.

The main program is starting here. User real and user imaginary. Okay. So we take the input from the user. For example, assume that.

```
29 * int main() {
30     double userReal, userImaginary;
31
32     cout << "Enter a complex number (real part): ";
33     cin >> userReal; ✓
34     cout << "Enter the imaginary part: ";
35     cin >> userImaginary; ✓
36
37     // Create a complex number using the user's input
38     Complex userComplex(userReal, userImaginary);
39
40     // Create another complex number using the constructor
41     Complex constructorComplex(3.5, 2.0); // Example values
42 }
```



So, 2 plus 3i is my input, user input. Right? So now, here you go. I have the user complex object. Right?

User complex object. I am passing user real and user imaginary. Right? So, whatever I have taken as input from the user, I am passing this. So this is the object, user complex object.

So that means this will invoke, right? Whenever you are creating an object, this will invoke the constructor, right? So, the constructor will be invoked. Suppose I am passing 2, 2 plus 3i, right? Yes.


So, real will be 2, and imaginary will be 3, right? In the first case. So, that means under user complex object, my real will be 2, and the imaginary part will be 3. And you have another object called constructor complex. So, in the constructor complex, you are passing 3.5 and 2.0.

So, one we said user has to give us the input, and another one you are passing. So, this is an example. So, 3.5 and 2.0, assume that 3.5 plus 2.0i. So, when this object is being created, again the constructor will be called. So, under this object name, what is the object name?

Constructor complex. So, your real will be whatever number you have passed, and imaginary will be whatever number you have passed. So, I passed 3.5 and 2.0. Alright. So, this is by user, and this is by constructor.

Right. I hope you understood. So, now we will proceed further. Alright. So now we are calling.

```
43 // Add the two complex numbers
44 Complex result = result.add(userComplex, constructorComplex);
45
46 cout << "User's Complex Number: ";
47 userComplex.display();
48
49 cout << "Complex Number from Constructor: ";
50 constructorComplex.display();
51
52 cout << "Sum of Complex Numbers: ";
53 result.display();
54
55 return 0;
56 }
```



Alright. So let us say the complex result. The result is an object. Alright. So, the complex result.

The result is an object. So that object is invoking add. Right. So that object is invoking add. So you are passing user complex and constructor complex.


Alright. So this is exactly what we are doing in the case of add. Complex C1, so now C1 is user-defined, right? User complex C1 is nothing but user complex, and C2 is nothing but constructor complex. So now here C1 and C2, the corresponding C1 real and C2 real will be added, right? User complex real part and user complex, right? Constructor complex real part will be added, similarly, The respective imaginary part will be added.

Now it is some real and some imaginary. Alright. So now it is returning complex. Alright. So which is carrying some real and some imaginary.

```

16 // Method to add two complex numbers
17 Complex add(Complex c1, Complex c2) {
18     double sumReal = c1.real + c2.real;
19     double sumImaginary = c1.imaginary + c2.imaginary;
20     return Complex(sumReal, sumImaginary);
21 }
22
23 // Method to display the complex number
24 void display() {
25     cout << real << " + " << imaginary << "i" << endl;
26 }
27 };
28

```



Alright. For example, if I am adding 2 plus 3i and 3 plus 5. 3.5 plus 2.0i. Alright. What will I get?


5.5 plus 5.0i. Alright. So this will be returned. In line number 20, right? So when it is returned, if you look at the main program, that will be stored in result, right? So that means result.real will be 5.5 and result.imaginary will be 5.0, right? So you can see over here, right? So that will be returned to result. So result carries the real part and imaginary part. So whenever I have a complex The object, so here the object is result.

So the result has real and imaginary. Look at line number 6 and 7. So that will be returned, and this result.add, when it is invoked, the addition will be returned to result. So the result has the real part and imaginary part. So now see how user complex number you are displaying.

```

43 // Add the two complex numbers
44 Complex result = result.add(userComplex, constructorComplex);
45
46 cout << "User's Complex Number: ";
47 userComplex.display();
48
49 cout << "Complex Number from Constructor: ";
50 constructorComplex.display();
51
52 cout << "Sum of Complex Numbers: ";
53 result.display();
54
55 return 0;
56 }

```



What is user complex number? So let us say 2 plus 3i. Right? Suppose 2 and 3 are the input. 2 plus 3i.

This will be displayed. And the constructor will be displayed. Now the sum of these two complex numbers. So that is why object-oriented programming will be elegant. Right?

So you have done everything in the class. Right? And then the member function is doing all the addition. The constructor is doing all the addition. The constructor initially saves, initializes, and then you have a member function that will do the addition.

So now here you have `result.display`. The sum of the complex numbers: `result.display`. So that means the result as the real part and the result as the imaginary part. So that will be displayed. Suppose, let us assume my input is 5.0 and 2.0 as the input.

User input. Right? So, we have already passed 3.5 and 2.5. That is a constructor. If you look at the program, we have passed this.

3.5 and 2.0. Right? 3.5 and 2.0. 5,2 is my input. User input.

User complex. Right? So, when I add this, 5 plus 3.5 is 8.5. 2 plus 2 is 4. So, I will get the output 8.5 plus 4i.

So, the addition of two complex numbers, right. So, the crucial part here is, when you are declaring the object, instantiate an object result, right. On the right-hand side, this object result is invoking the add member function. So, you are passing user complex. So, this will be

Considered as C1 inside the function, and this will be considered as C2 inside the function, right? Copying the usual one, call by value, call by reference, alright. So here it is C1, you are passing right, user complex, and you are passing constructor complex, right. So, when you are passing this, this will be assigned to C1 and C2; it is exactly happening over here. And then, the corresponding reals will be added. Corresponding imaginary will be added.

And line number 20. It is returning the complex object. Alright. Which is carrying some real and some imaginary. Alright.

So it is nothing but. You see it is a special member function kind of. Right. Complex, some real, some imaginary. Right.

```

16 // Method to add two complex numbers
17 Complex add(Complex c1, Complex c2) {
18     double sumReal = c1.real + c2.real;
19     double sumImaginary = c1.imaginary + c2.imaginary;
20     return Complex(sumReal, sumImaginary);
21 }
22
23 // Method to display the complex number
24 void display() {
25     cout << real << " + " << imaginary << "i" << endl;
26 }
27 };
28

```



So here you have. When you look at the constructor complex double R double I, R will be assigned to real and I will be assigned to imaginary. This is what exactly happening. Alright. And then where is it returning?

You can see that it will be returned to the result. So that means the result has real part, result has imaginary part. So that is what exactly we are displaying in line number 52 and 53, okay. So this is how this particular program is working using parameterized constructor. So you can do the same problem in one more way, right.

So, how to do that? The same, right? So, we have real and imaginary, and we have this constructor. It is nothing but the parameterized constructor. Up to line number 15, it is the same.

And here you go. So, here we have complex add. Right. I have a complex reference variable, other. Right.

It is exactly like a few classes back we have done. Right. Addition of two distances, if you remember. Right. Feet and inches.

So here you have real with the other real. Imaginary with the other imaginary. So, we will see how this will be working, and the rest of them are the same. We are going to return some real and some imaginary, whatever we had seen in the previous program and the display function, right. So, only this part is crucial.

We are going to see how this is working. So, now the user real and user imaginary, right. So, when you are creating this particular object. The parametric

constructor will be invoked with the user real and user imaginary. And then another one is the constructor complex.

You are passing the values 3.5 and 2.0. Exactly the same values we are passing. Only the line numbers 16 to 21. All right. So it is a different way.


In fact, we had seen this. I will explain one more time for the other example. So now you can see this. All right. You have declared complex result.

Now the user complex. The user complex is invoking the add member function. And you are pausing the constructor complex. Right. User complex is invoking the add member function.

And then you are pausing the constructor complex. So, if you look at the syntax, it goes like this: complex and other complex reference other, which will be equal to right. So, complex and other correct, which will be equal to constructor complex, right. Right. So the meaning is other and constructor complex are pointing to the same location.

```
43 // Add the two complex numbers
44 Complex result = userComplex.add(constructorComplex);
45
46 cout << "User's Complex Number: ";
47 userComplex.display();
48
49 cout << "Complex Number from Constructor: ";
50 constructorComplex.display();
51
52 cout << "Sum of Complex Numbers: ";
53 result.display();
54
55 return 0;
56 }
57
```

*Complex & other constructor complex;*



Right. You are copying the reference. Right. Pass by reference. Right.

So user complex is invoking the add function. You are passing the constructor complex. So complex reference operator other. So both are pointing to other which is available in your add function, right? That is also a complex object, right? Complex class other object is a complex class and constructor complex, which is also a complex class. So they are going to point to the same location, right? So the content will be the same. So suppose I have what is that constructor complex your 3.5 comma 2.0 means other will also have now 3.5



comma 2.0, okay? That is the meaning. So since here your user complex is invoking the function, right? Since the user complex is invoking the function, your real user complex real, whatever you are writing as an input, giving as an input, user complex, right? So that will be your actual real and the imaginary the user complex, since it is invoking So the user complex real will be real here, and user complex imaginary will be imaginary here. And then the rest is the same. You are adding the corresponding real part, and you are adding the corresponding imaginary part. So then you are returning.

Now, the rest of the things are the same as whatever we had seen in the previous program. All right. When you run the code. All right. So now, everything is the same.

So only here is it crucial. Line number 44 is crucial. User complex is invoking, so that means user complex dot real will be the real that you are using in the function, and user complex dot imaginary will be the imaginary, right? So now, you display the user complex numbers, constructor complex numbers, and the sum of these two. So when I run the code, assume that 5 comma 2 is my input.

Right, assume that 5 comma 2 is my input, and 3.5 comma 2.0 we are passing, all right. So we will get the output 8.5 plus 4i, all right. So 5 plus 3.5 is 8.5, and 2 plus 2 is 4, right? So this is how we can have the usage of a parameterized constructor, right? So in one case We have done. The result is invoking. The add function. All right.

You are passing two arguments. C1 and C2. User complex. And the constructor complex. Correct.

And then it will be copied. To C1 and C2. And then the rest is the same. The corresponding. The real numbers.

The real part will be added. Real parts will be added. And the corresponding imaginary part will be added. So the complex addition. Has been done.


So this is the case. When. The object result itself. Is invoking add. So in another case.

Here you have. The user complex object. Is invoking. The add member function. And then you are passing.

Constructor complex object. So when you are passing the constructor complex, the user complex dot real will be the usual real in the function, right? That is line number 18. And user complex imaginary will be the imaginary in the function, right? So anyway, the other will be copied, right?

```
43 // Add the two complex numbers
44 Complex result = userComplex.add(constructorComplex);
45
46 cout << "User's Complex Number: ";
47 userComplex.display();
48
49 cout << "Complex Number from Constructor: ";
50 constructorComplex.display();
51
52 cout << "Sum of Complex Numbers: ";
53 result.display();
54
55 return 0;
56 }
57
```

*Complex & other constructor complex*



So that means this constructor complex will be copied into other, right? So other.real and other.imaginary. So the corresponding real part and the corresponding imaginary parts will be added, and then that is returning some real and some imaginary. So when we are running the code, alright, so we are getting the same output. So this is how we can use the applications of the parametrize constructor.

So we have seen several applications of the default constructors and the parametric constructor. So, in the parametric constructor, this particular problem is slightly interesting because the tool input, that means the real and imaginary, you are taking from the user, and another one you are passing as a constructor. So, slowly you are getting familiar with these object classes and the constructors concept. Right.

Apart from that, we have also seen if it is possible to define the member function outside the class. So, in a similar way, so the special member function constructor, you can think of. Whatever program that we have written so far, so is it possible that we can write this constructor outside the class with the syntax that we have used?

So the next concept we are going to see is the copy constructor. Right. So what do you mean by a copy constructor? So here. Let us assume we have a class example, right?

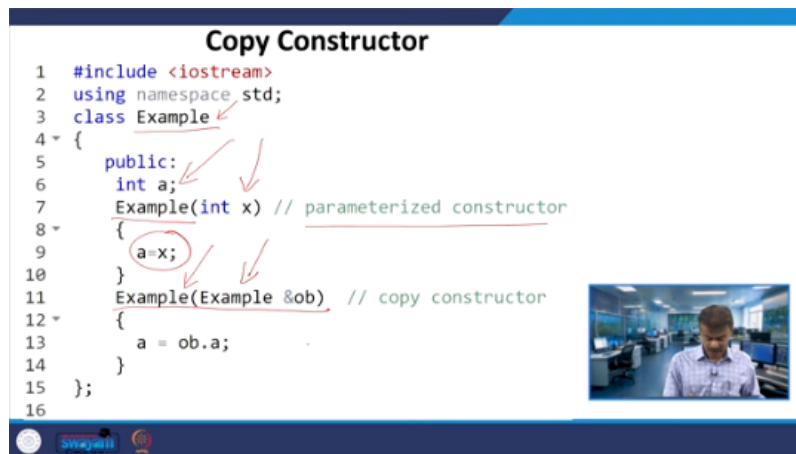
So let us assume we have a class example. I have one member data int A, right? So I have one member data int A, right? So here I have one parameterized constructor, the usual one. Parameterized constructor, now you know.

Example, right? So which is the same as the name of the class, right? Which is the same as the name of the class. And you are passing one parameter. Int x. Right.

So you are passing one parameter. Which is int x. So here you have. It is a special member function. So I am copying x into a. That means a equals x. I am copying. x into a. So a equals x. And now here.

I have another constructor. So this is the copy constructor. Right. So here you can see the name of the class example. And I am using object reference.

Right. So example is a class. And here the object reference. Address OB. Right.



And OB.A. The object.A is equal to A. Right. So, in fact, A equals OB.A. Right. OB.A is assigned to A. I mean to say. Right.

So A equals OB.A. So, this we call it a copy constructor. So, we will see in the main program how we can use this. Right. So, here you go. Your class is over here.

Line number 15. Your class is getting over. So now, this is the usual one. When you are calling. When you are creating.

The object E1. Alright. So, you are creating an object E1. By passing the value 36. Alright.

So, what exactly is happening? In line number 19. Alright. So, that means you are calling. The parameterized constructor.

Right. You are creating an object E1. Passing the value 36. Alright. All right.

So, the meaning is you are invoking. All right. In fact, this will be the E1. Right. When you are creating E1.

So, it is automatically invoking the constructor. So, here in this case, the parametric constructor because you are passing the value 36. Right. So, that means A is taking the value 36. So, here you are passing.

So, int x. So, x is 36. So, 36 will be copied into A. So, A equals 36 for the object E1, for object E1, right. So, it is nothing but E1 dot A, which is equal to, suppose I am accessing this from outside, right. In fact, it is outside. E1 dot A is 36, right.

So, the next line is slightly different. The first time we are seeing you have the class example, and then you have object E2, right? So, object E2, you are passing another object E1, right? What is the meaning of this? E2, you are passing another object, right? So, that means your example is the class, name of the class, address OB, which is equal to what you are passing E1, right?

So, when you are creating this object, E1 will have some memory, right? So, let us say this will be the location 400. When I display E1, see out E1, so it will give 400, location 4002. So now when I have this, example reference operator OB, whereas if you look in the main, I am passing E1. I am passing E1.

So when I am passing E1, right, so here you go OB and E1, which is pointing to the same location, right? OB and E1, which is pointing to the same location, right? So, we know E1 dot A is what? 36. E1 dot A is 36. So, OB is pointing to 4002, E1 is also pointing to 4002. Assume that this is some hexadecimal number. Correct? So, E1 dot A is 36, that we know already. Right?

So now if E1 dot A is 36, what is OB dot A? That is also 36 because both are pointing to the same location. Correct? OB dot A is 36, so that you are assigning to A. That is the meaning. OB dot A is A. Right?

So now when I want to print E2 dot A, alright? So that means when I am creating an object, OB dot A, you are passing, right? Assigning to A means your E2 dot A, which is also 36. That is the meaning, alright? OB dot A, you are assigning to A. OB dot A is what? 36. So here when it is equal to, right? A equal to OB dot A, so A equal to 36 for object E2. Otherwise, you call it as E2 dot A is 36. Alright.


E2 dot A is 36. So when I want to display this line number 21. See out is E2 dot A. So you know the output. So what will be the output we will get? We will get the output 36.

**Copy Constructor**

```
1 #include <iostream>
2 using namespace std;
3 class Example
4 {
5     public:
6     int a;
7     Example(int x) // parameterized constructor
8     {
9         a=x;
10    }
11    Example(Example &ob) // copy constructor
12    {
13        a = ob.a;
14    }
15 };
16
```

Handwritten annotations and diagrams:

- Object  $e_1$  is shown with  $e_1.a = 36$ .
- Object  $e_2$  is shown with  $e_2.a = 36$ .
- Object  $ob$  is shown with  $ob.a = 36$ .
- The assignment  $a = ob.a;$  in the copy constructor is circled, with a note  $a = 36;$ .
- A note  $Example \& ob = e_1$  is written below the copy constructor.
- A diagram shows a box labeled  $e_1$  with an arrow pointing to  $ob$ , indicating that  $ob$  is a reference to  $e_1$ .



Alright. I hope it is clear for everyone. And line number 20, if you look. So, this is called the copy constructor. First time we have seen.

We have seen the default constructor. We have seen the parametric constructor. And this is the syntax for the copy constructor. So, object E2 is an object. And you are passing an object.

That object is E1. You are passing the object E1. So, and then once you are passing this. So, this is the syntax. So, example address OB, which is equal to E1.

Right. Example address OB, which is equal to E1, and then if you look over here, both OB and E1 are pointing to the same address. So that means E1.A is 36, and

OB.A is also 36. Right. So when OB.A is 36, and you are assigning to A, this will be nothing but E2.A.


Alright. The meaning is this is nothing but E2.A. What is the value of E2.A? 36. So when you are trying to display this in main, E2.A, you are getting 36. Alright.

I hope it is clear for everyone. And the last concept that we are going to see in C++ is the destructors. Right. In today's last class. So let us consider the class employee.

```
17 int main()
18 {
19     Example e1(36); // Calling the parameterized constructor
20     Example e2(e1); // Calling the copy constructor
21     cout<<e2.a;
22     return 0;
23 }
24
```

stdout

36



Alright. So here you have the default constructor. Alright. So, in the default constructor, you are displaying 'constructor invoked.' And then, for the first time, we are seeing this as a destructor.

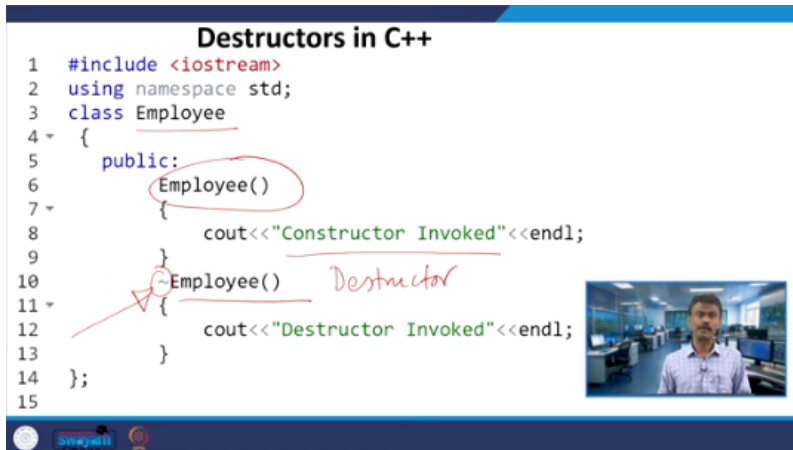
Alright. So, this is a destructor. So, the destructor symbol is a tilde, and then you have 'Employee.' Same. Almost like a constructor, but the only difference is you put the tilde symbol.

So, that is nothing but a destructor. When you are creating memory, when you are creating an object, you require the constructor for the memory, and then you have to destroy the memory. So, for that, you require destructors. So, you might have used the 'new' operator and 'delete' operator in the case of an array. So, the 'new' operator is for allocating memory, and the 'delete' operator is for destroying the memory, de-allocating the memory.

So, for the deallocation, we have the destructors, right. Assume that you have an object E1 whose class is Employee, right. So, you are just printing, see out

address E1. I am printing the address of E1, right. I am printing the address of E1.

And another object E2, I am printing the address of E2. So, when you are creating an object, the constructor will be called, right. When you are creating the object, the constructor will be invoked will be displayed. Right.



So, the constructor is invoked. I will just simply write CI. It stands for constructor invoked. And the address of E1. It will print something.

Number. Right. Extra decimal number. Address of E1. And employee E2.

The constructor will be invoked. And address E2. Some other address. 0x23E7. Something like that I am writing.

Some other address. And then. Return 0. So before that, you will have the destructor. That means you are deallocating the memory for E1 and E2.


Right. So, two more outputs you will get because of two objects. So, you will get the destructor invoked. Right. So, I will write DA two times.

Continuation DA. DA stands for destructor invoked. CA stands for constructor invoked. So, this you have to get these four as output. So, you can see here.

Constructor invoked. Whenever you have employee E1. Then you have address E1. Employee E2. Constructor invoked.

```
16 int main(void)
17 {
18     Employee e1; //creating an object of Employee
19     cout<<&e1<<endl;
20     Employee e2; //creating an object of Employee
21     cout<<&e2<<endl;
22     return 0;
23 }
```

Constructor Invoked  
0x7ffddb0c06  
Constructor Invoked  
0x7ffddb0c07  
Destructor Invoked  
Destructor Invoked



And then you have address, and finally, before going to the return 0, both E1 and E2 because of this tilde employee, the destructor. So what you can do is you can try without line number 10 and 13, right? Without line number 10 and 13, you try the same program; the last two outputs will not be displayed, right. So this is the Usage of the destructor; that means now this memory space, right? So these two will be deallocated; the objects E1 and E2 are deallocated from this memory space. So that is the meaning, right? That is what you are getting as an output. So in today's class, we talked about the parameterized constructor.

And then the applications that we had seen in terms of the addition of two complex numbers. We had also seen one more constructor called the copy constructor. And now you know the meaning of the destructor as well. Thank you.