Programming in C ++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
Copy Constructor and Copy Assignment Operator (Contd.)

Welcome back to Module 14 of Programming in C++. We have been discussing about copy construction process, we have introduced what a copy constructor is. It is a special constructor which takes an object and makes a copy of it.

So, in that process, it copies each and every data member in whatever way the programmer has decided to do and we have explained that the copy construction is extremely necessary for the purpose of call by value, return by value mechanism and for initializing data members which are part of another object which is being copied. So, if you copy an object, you need to copy its data members. So, you will again copy constructor for that.

(Refer Slide Time: 01:10)

So, now you will take quick examples into this. So, again back to our complex class, there is nothing different here except we have added a function display which takes a

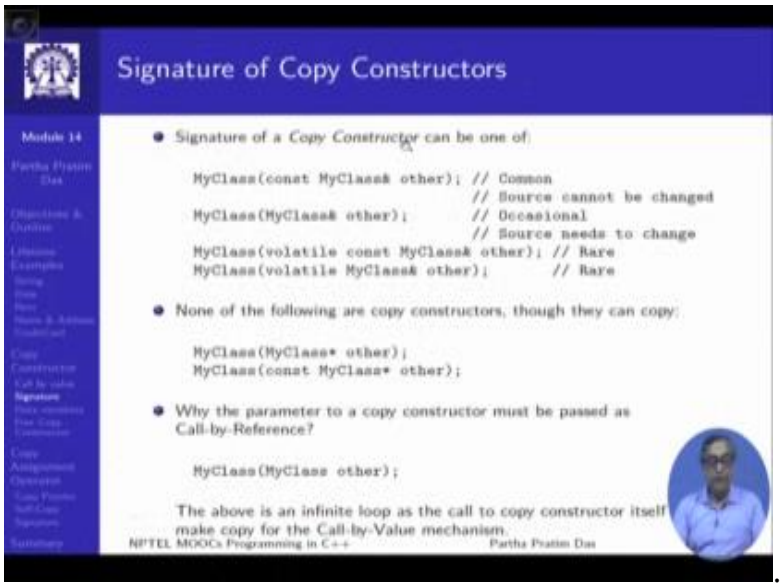
complex number here and prints it and the wave we have designed, this display function is the complex number is past as a value. So, it is a call by value. So, now if you look into the order in which the constructors are called naturally, first this will get executed. So, the constructor gets invoked which is this constructor, these output.

Then, once C has been constructed, it has to come here which mean that before the function can be called this actual parameters, C has to be copied to this formal parameter c param and this process is the copy construction process. So, at this point the copy constructor that is this constructor will get invoked and you can see that it is printing the message copy constructor. In this case, it has the same set of member values because that is how it has been set, but it is a different constructor from the original constructor of C. Then, the display happens and that is the function executes. So, it prints. So, this is what you get to see and then, the function is making an exit from this point, there is a control. We will have to now come back to here.

When this happens naturally, let me clean up. When the function is ended, the scope of this particular parameter, the formal parameter which is like a function local ends at this point, this also is an automatic object and since it ends at this point, naturally the objects c param that was constructed. So, the c param, this object will no more be available after the display function has returned a control to main.

Therefore, it has to get destructed. So, now, it will get this call of the destructor for c param. Then, the control comes here and then, we are about to return and when I reach the end of the scope of main, the destructor for C gets call and this is how the order goes. So, you can easily see that whenever you will do call by value, you will be able to see such copy construction and the corresponding distraction of the object which is a formal parameter to happen.

(Refer Slide Time: 04:12)



Let me specifically talk about the signature of copy constructor as C++ as a well specified signature of a copy constructor. This is the signature that is most commonly used. So, I have highlighted it here. This is a common one which takes the object as a call by reference, as a reference object. You can see the reference here. Since it is taken as a reference as you know, it can be changed because it cheers the same memory location as of the actual parameter that we put a const. So, you are saying if you are making copy, then whichever we are coping from that object cannot be changed.

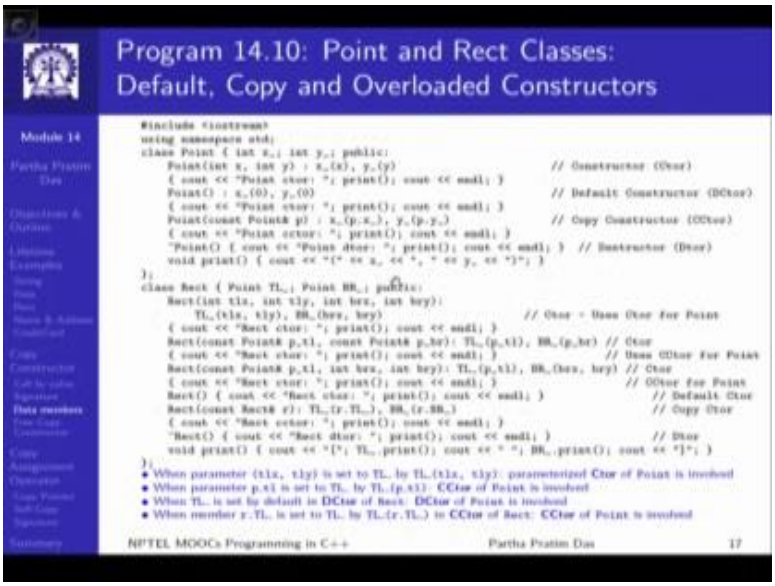
So, that is what is clearly specified by this signature and that is the most commonly copy constructor that will always see, but occasionally we may drop this const and have a copy constructor like this, where we are saying that while we are coping, it is possible that we also change the object that we had copying from. I am sure this will sound very weird to you right now, but please bear with me. At an appropriate point, I will show examples of why having this features is very important for certain designs and then, there are certain other copy constructors which use volatile data. So, those are specified here.

They are very rarely required specifically if you are using embedded system programming. So, you may look this up when you are doing that kind of a programming and also, you please note that some of the similar signatures like I could have past the

object, I want to copy from, I could passed pointed to that object or pointed to the constant object and so on. These C++ do not recognize them as copy constructors. So, if I provide them, then they will be taken as just another constructor, but they will not be invoked at the time of call by value. So, this will have to be known in mind and a final observation is what if I wrote my copy constructor like this and what if I passed the object to copy from as a parameter to the constructor copy, constructor with call by value mechanism and not by call reference.

I can see that this is not going to work because a copy constructor is also a function. So, if we take a parameter other at the time of call by value, then this parameter will also need to be made available to this constructor. So, this itself is a call by value which will mean that this will again call the copy constructor and to call the copy constructor, it needs to do call by value. To do this call by value, it needs to call the constructor. The constructor needs to be called by call by value and so on so forth. So, it simply keeps on going. This eventually turns out to be an infinite loop and it could not have worked. So, you cannot pass the object to a copy constructor by call by value.

(Refer Slide Time: 07:50).



Let us look into some bigger examples. These are naturally little bit longish code. So, these are more for you study than the discussions here. I will just outline what I am

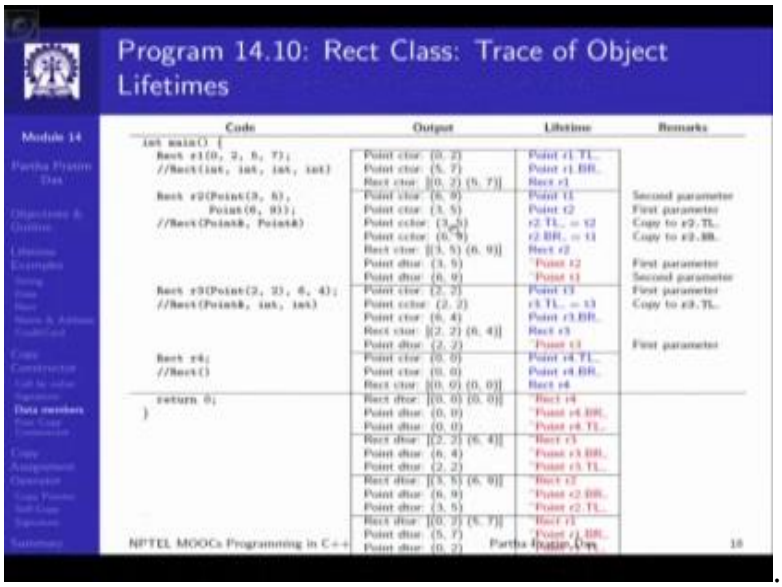
trying to demonstrate here is we are showing default copy and overloaded constructors here. So, there is a constructor, there is a copy constructor, there is a default constructor for the point plus that we have seen earlier, then we have several constructors for the rectangle class, we have 5 constructors.

The first point takes four points, the second one takes I am sorry, the first one takes four integers that is like why coordinates of the two corner points, the second one takes two different corner points, the third one takes a point and coordinates of the other point and then, last one there is a default one and the last one is a copy constructor.

So, you can try to see, you can try to convince yourself studying all of these constructors and their initialization list as to which constructors will get called. For example, I would just focus on this, the constructor which takes two points. Now, naturally that constructed needs to take to point T l and copy it to the T l members. So, what I want to do is basically take this and copy to the T l member and that is what is written here. How I do it? Naturally to be able to do this, I need the copy constructor of point. I cannot do it with a parameterize constructor or a default constructor of point.

These are not useful because I already have a point and I want to initialize another point. So, this is the point that I was making that when you have data members of user defined type, and then you will require the copy constructor for those data member types to be available, so that you can easily copy them. So, if you do not for example say point class. If the point class does not have a copy constructor, then this code or this code will not compile because you will not be able to copy the point as it is required here.

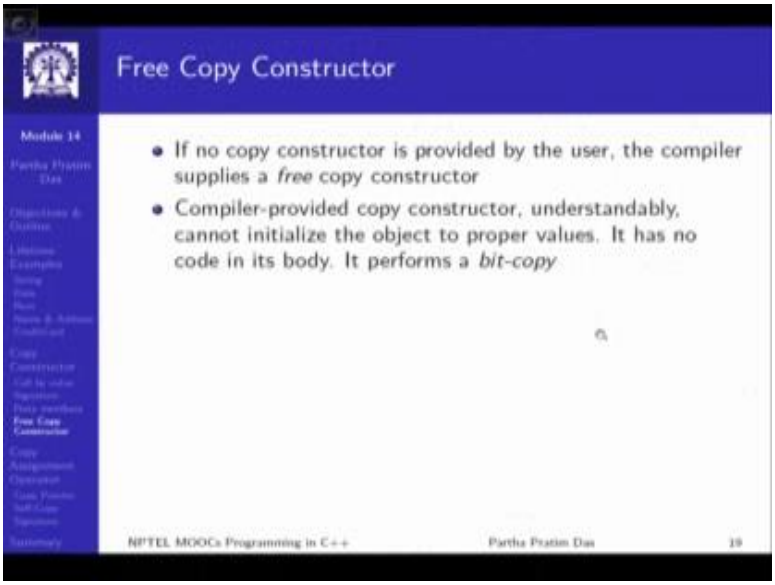
(Refer Slide Time: 10:34)



So, this is just an small application between here with different rectangle objects being constructed and subsequently being this destructed and if you look into the table here, we have shown all the different objects that are getting constructed in this process and the life time of which object longs for how long. Again the idea is not to explain it here, but I have worked this out, so that you can take this code carefully, study and convince yourself that you understand exactly how the construction, the copy construction. You will see several copy constructions happening here. You can convince yourself as to why this copy construction happens.

For example, if I construct a point and pass, it will construct a copy. In our case, it will not because the point is passed as a reference, but when I set the TL field or the BR field of a rectangle object, then it certainly will lead the copy construction to happen. So, please go through this and convince yourself that you understand the whole process of construction, destruction and copy construction together.

(Refer Slide Time: 11:53).



Now, like for the constructor and destructed, we have free versions. The same mechanisms are also available for copy constructor. If no copy constructor is provided by the user, by the programmer, if I write a class which does not have a copy constructor, the compiler will supply a free copy constructor and the compiler provide a copy. Constructor certainly cannot initialize the object because it does not know how to do it. So, what it does is, it simply makes a bit copy which means it takes the complete bit pattern of the object to copy from and makes another bit pattern exactly seem as the object.

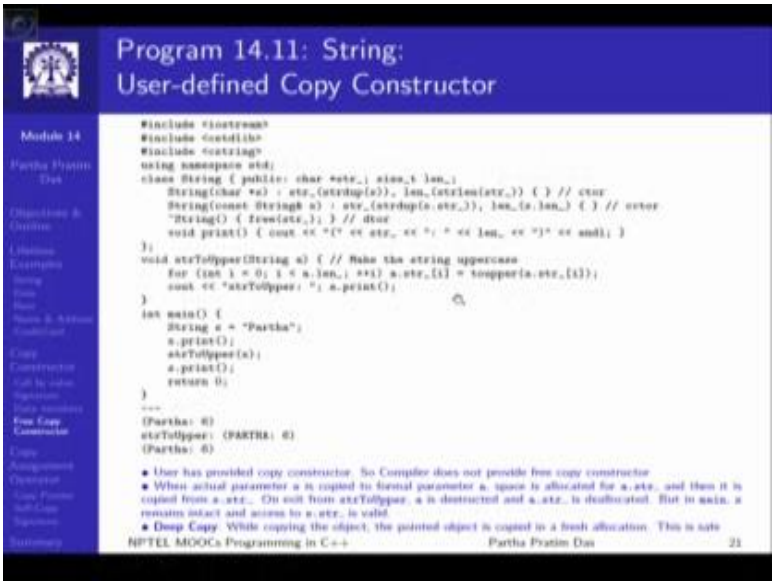
Now, you will ask what is wrong in that. That should actually generate a copy. Now, that is where we get into severe problems. We will just illustrate soon that times just copying the bits is not copying the object, but if it is same that is if it is copying the bits is copying the object, then we can leave the free copy constructor that the compiler provides and the story is similar. If we provide a copy constructor, then the compiler stops providing one.

(Refer Slide Time: 13:10)

So, we are back to the complex class. We are now using the free copy constructor. So, the copy constructor that was written, I have simply made them commented and I am still trying to use the call by value calling the display function and you will see if you compare with the previous one, this is the one which you have seen earlier and this is the one when we provided the copy constructor earlier. This is the one here where there is no copy constructor given the compile is providing that and the only difference is since no copy constructor is given by us, there is no explicit message saying that the copy is being constructed, but otherwise if you look at the output, it is exactly the same.

So, it does not make a difference which means in other terms in this case copying bits is same as copying the object, but do not generalize. Hold on, do not generalize.

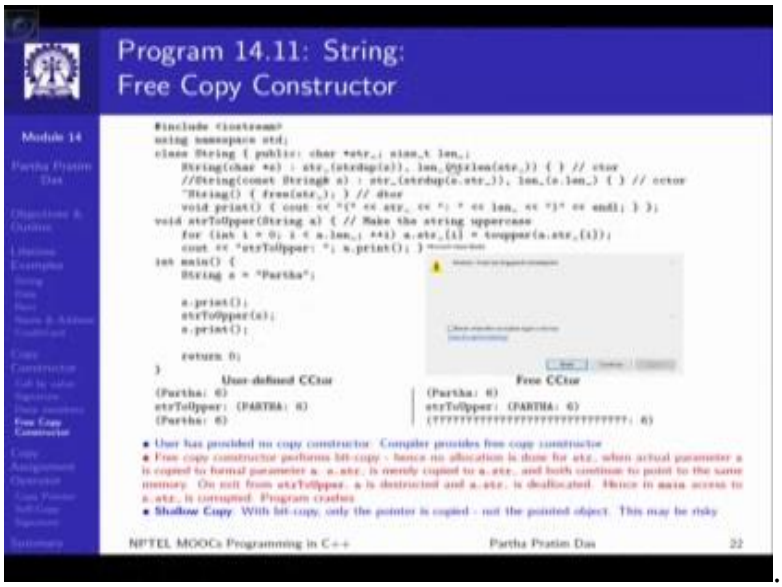
(Refer Slide Time: 14:16).



Let us consider, let us go back to our string class and we will try to copy. So, if you try to copy, I have written a copy constructor, simple. What the copy constructor will have to do? Simply you will have to make that copy of the string. So, it does another strdup on the string member of the S object, the object to copy from and it copies the length that will suffice in copying the string.

Given this copy constructor, we have also written a simple function which takes a string and converts to upper case. So, I construct a string and print it, you get to say this, then we call strtoupper, then print it again and this is the output that will expect which is pretty much fine and note that in string, we have actually pass this as value. In strtoupper function, we have passed this as a value.

(Refer Slide Time: 15:37)



Now, I do the same thing, but only thing is I comment out the copy constructor. I have commented out the copy constructor. Unfortunately for our convenience, the compiler will not complain. The compiler will say it is fine. You have not provided a copy constructor. The free copy constructor would be there which will copy the bit pattern and therefore, the same program will compile correctly and this is the output with the user define copy constructor, but when I run this program, I get this kind of an error. This is fine; this is fine - but just look at these point.

I get some garbage output and then, the program crashes. Now, you need to understand a little bit in terms of what is happening here is if you look into the original object, this is how the original object looked. So, this is what goes to Partha, this is the STR and it goes to Partha. This is len which is x.

Now, since I do a bit copy, so I get another object which has exactly the same set of bits which mean it has exactly the same pointer address as in the original one. So, in my copied object also, I have the same values 6 for length and I have the same object we used here. When I called the upper, I made this. So, this was S. I am calling the upper here. This is S and the formal parameter is A. So, this is A, and why did it copy like this

because I have not provided a copy constructor. If I had done one as I did before, then it would have looked like this S which has Partha 6.

Then it is A which has Partha because I had explicitly deduct strdup of this string into the new object string, but because I did not provide a copy constructor, the bit patterns I have got copied. So, naturally pointers are same. They are pointing to the same thing. So, this is what happens when I am here. When I have entered the function strtoupper, naturally it has a valid string. So, it will take things to upper case, it takes things to upper case, prints it, fine.

What happens at this point is, these are point where the scope of strtoupper ends. We know that this is an automatic object and its scope ends at this point, so that distractor of string gets called. What does a distractor do? It freezes string. So, at this point what happens is the distractor of A is called. So, this will free up this string. So, this memory goes and I am back here at this point. I am back here. Now, what happens if this memory has gone? Then for S, the STR now points to nothing. Whatever it used to point to that address is with us, but at that place, there is no string available because I have already free that to the system disaster to say the least.

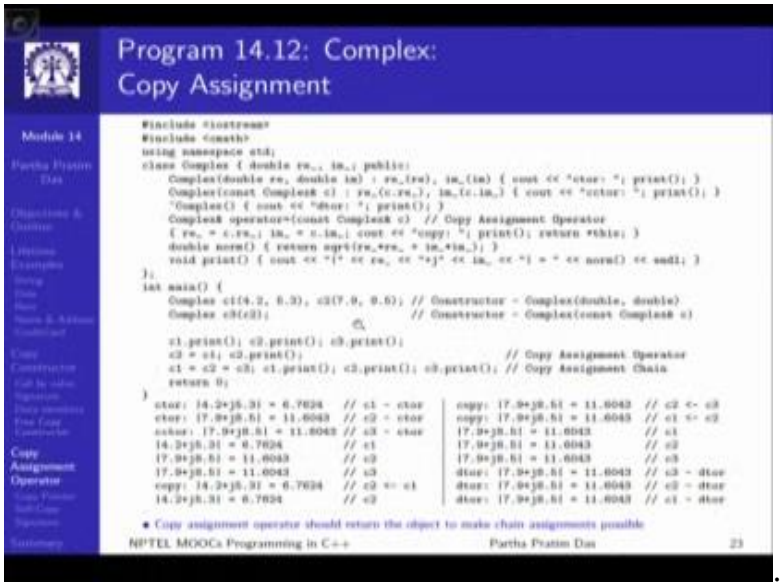
That is the reason when I am trying to print them, it prints some garbage characters. It is not necessary that it will print question mark. I have just put question mark as a place hold. I can print anything because it just does not know where it is printing from and it results into this kind of an error of the program crush.

So, this example clearly shows you that bit copy is not same as object copy and that gives us two different notion of copy. When an object has pointers that is referring to other objects that are dynamically created, then while copying it if we just copy the pointer, but do not copy the object, then we say that we are doing a shallow copy, but in contrast while copying the object if we do not copy the pointer, but you copy the pointed object, then will say that we have done a deep copy.

So, while we wrote the copy constructor for the string, we did a deep copy which was fine because when strtoupper is called by deep copy, a different STR data was created

and that created STR data got destructed when strtoupper function finished, but when we did not do that, when we allowed the compiler to provide the free copy constructor, then it resulted in a shallow copy. So, when the local parameter or the parameter of strtoupper was destructed, the original actual parameter also lost this value. This is the kind of problem that you can get into if you do not distinguish between shallow and deep copy.

(Refer Slide Time: 21:44)



The next that we want to look at is known as copy assignment. In copy assignment let me first show you the example which we are I am sorry that is some objects have been created. These objects are created directly. C3 is created by copy construction. We have printed the so far is what you already know, but then now what you are trying to do is to assigns C1 into C2, that is this is what for built in types. We also make copies in object terms. This needs to be distinguished from the copy construction because when I write this C2 is being copied into C3, but with the fact that C3 does not exist, C3 has to get created and while it gets created, it must be a copy of C2 whereas if you consider this assignment, C1 being assigned to C2 then, C2 already exist, C1 also exist.

Naturally I want to take C1 and make C2 a copy of C1. So, both cases have copy, but with a fundamental difference that in this case, there was no earlier object and the object is created by copy and this case the object existed and we are nearly changing the data

members of the object from taking values from another object of the same time. So, this we said is the copy construction. This we say is the copy assignment or in some cases we said this is a simple assignment.

So, naturally if copy assignment has to be done, then we need this kind of an operator to be present. This is known as the copy assignment operator. If you recall the operator over loading discussions earlier that we have discussed that every operator has a corresponding operator function and the assignment operator has this function because what I am assigning, I am assigning C2 being assigned C1. So, this is equivalent to saying that C2 dot operator assignment C1. So, the function corresponding to the assignment operated is this function. It takes C1 which necessarily is an object of the same time and what does it certainly test an object of the same type.

Why certainly is questionable? It could have returned nothing also as long as it does the copy, but we will discuss why it should also return an object of the same type and in the process, it will copy the respective data members as we want.

So, whenever we write this, this operator will get invoked and according to this operator, the assignment will happen which will mean that in this logic I can decide what I want to copy. So, it is notionally a copy always means that it is kind of a clone, right. We loosely mean that it is a clone that it is identical to whatever I am copying from, but in actual terms of C++, a copy is not necessarily a clone. A copy is whatever I want to copy. I may not want to copy everything, I may want to copy some part, or I may want to do anything because I am actually being able to write an operator function for the copy.

Now, to answer the question it is easy to easy to understand that why the parameter to a copy assignment operator has to be of the same type as of with glass. So, it has to be complex. It is easy to understand as to why this should be reference because if this were not a reference, then you will unnecessarily require a copy construction to happen and then the assignment will happen.

So, that is not what you would want to do. It is understandable that if it is a reference, why should it be constant? It should be because while you are copying, you do not want

the right hand side to change. That is a common semantics. So, this part is clear. What is not obvious is why this return an object of the same time should? For that, just consider the next line in the example. What is C1? What is C1 assigned C2 assigned C3.

Please recall your C. This is a case where the associativity is right. It means that it happens from right to left. So, this is equivalent to C1 assigned C2 assigned C3. This is equivalent value which means the assignment of C3 to C2 must be an expression always as a value, right. It must be an expression and it must be such an expression, so that I can assign it to C1 which means whatever is the written type here that must be able to go as a parameter type of the same operator. If i do not ensure of that, I will not be able to write this kind of a chain assignment. I will be able to write one level of assignment C1 assign C2.

If operator assignment simply returns a wait or something else, then I will be able to still write this because by this I have made changes to the object from C1 from the object C2. So, it will give me the same effect, but I will not be able to write this chain assignment. That is a reason the copy assignment operator always has the same type. That is a reference to the class as input or parameter and reference to that class as a return type for the output, so that you can make this kind of a change assignment possible.

Now, having said that if you now go through this, we can quickly go through; these are three constructions. These are the normal constructions and these are copy construction, identical object coming from here. These three are the three print statements are from here. This is making a copy.

In the copy assignment operator, we have specifically written what copy clone, so that you can know that this is what is happening from your copy assignment operator. So, this makes a copy of C1 to C2 and then, you can see the print shows what is C2 and then these two where first C2 gets assigned to 3, then the result gets assigned to C1 and then, they prints and naturally the reverse order of their destruction. So, this will clearly show you that process of copy assignment for different objects in a class.