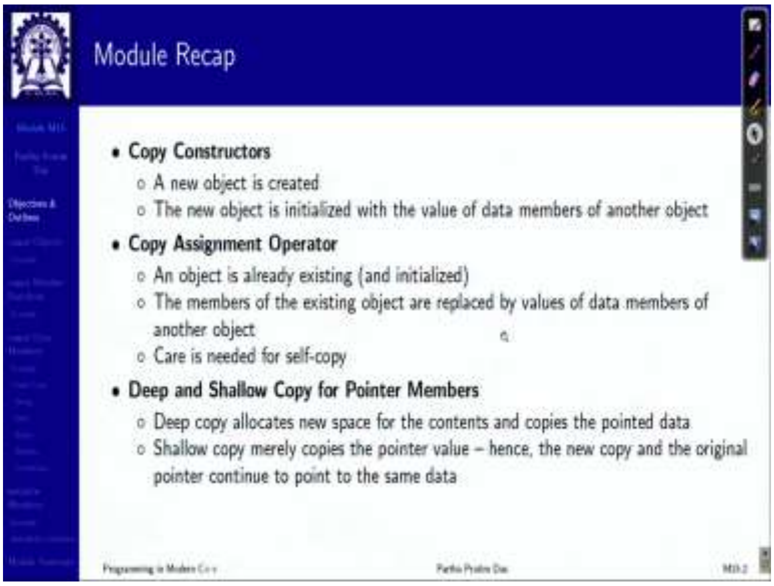
Programming in Modern C++
Professor Partha Pratim Das
Department of Computer Science and Engineering,
Indian Institute of Technology, Kharagpur
Module15 Lecture 15
Const-ness

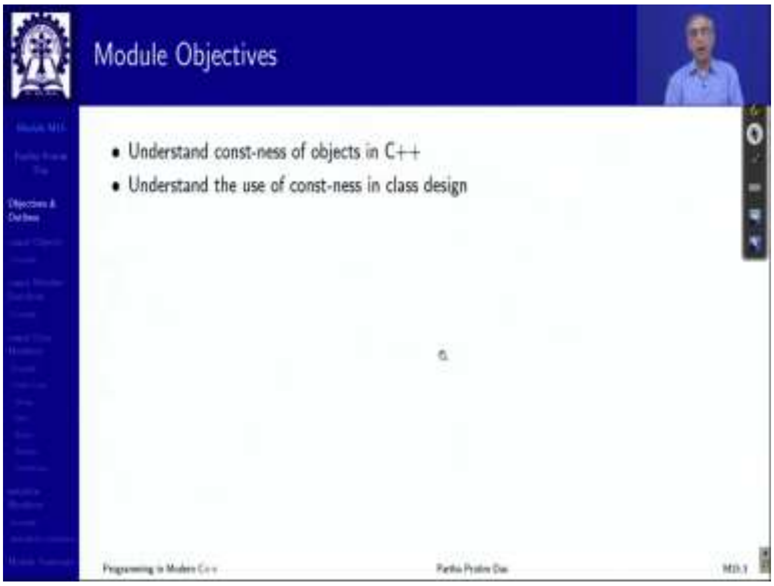
Welcome to programming in modern C++, we are in week 3 and going to discuss module 15.

(Refer Slide Time: 0:35)



In the last module, we have introduced very critical concepts of copying, as a construction when the object does not exist, and copying is an assignment when the object exists and being overwritten. And in this context, we have mentioned or discussed about deep and shallow copy issues, which are very critical.

(Refer Slide Time: 0:54)

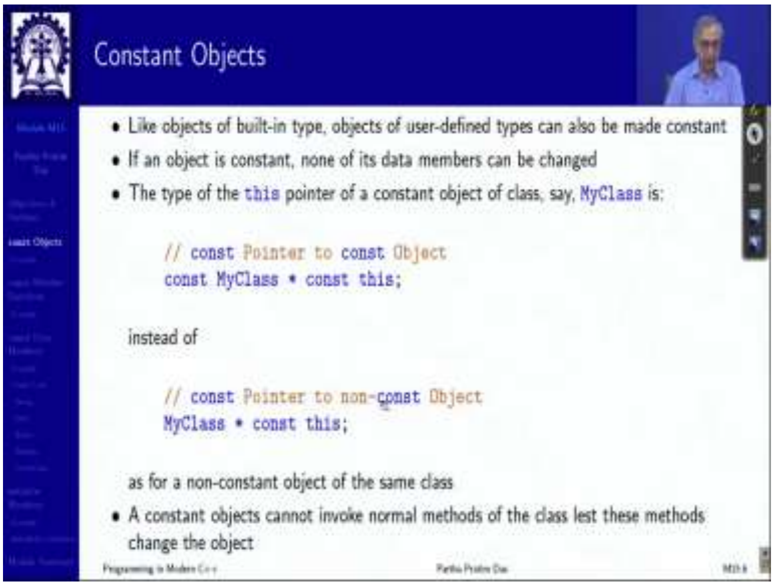


In this module, we will look at the effect of const-ness in the design of the user defined types, we have seen const-ness in the context of built in types. What does const mean? What can you do what is a constant pointer and all that.

(Refer Slide Time: 1:15)

We will see the consequence of all that, in terms of the objects data members and its member functions, that is the basic objective.

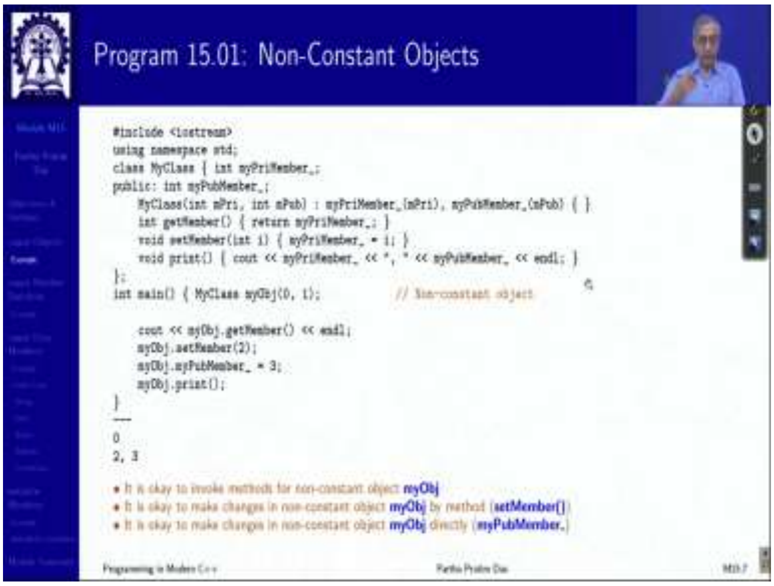
(Refer Slide Time: 1:22)



So, a constant object is an object which cannot be changed like user defined types, any object can be made constant, and an object is constant simply means that it state cannot change, nothing can change in that object in terms of the data members.

So, when you do that, what happens in the this pointer type of the this pointer in the for that particular object changes. So, earlier this pointer was it was just a constant pointer, because you cannot change the identity, but now, you have put you have a const in front of this, which say that the object pointed to cannot be changed. So, nothing can be changed in that object. That is the simple extended idea of constant object based on the…

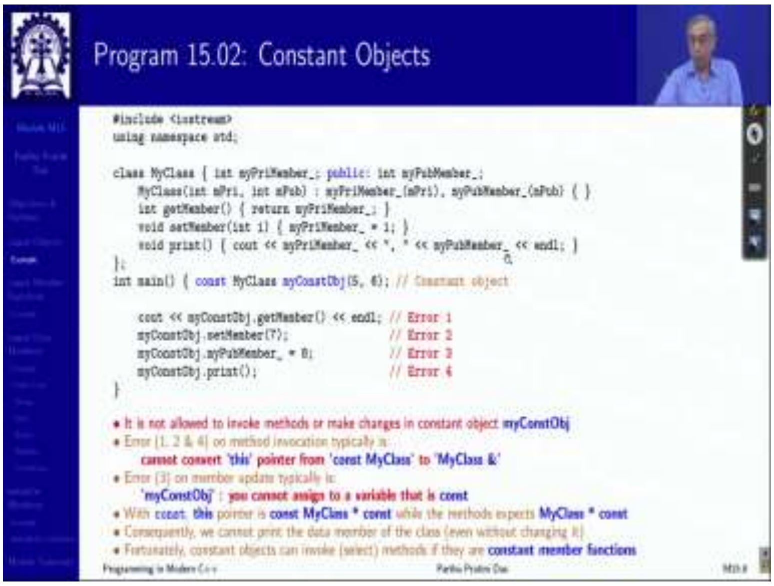
(Refer Slide Time: 2:16)



So, here is a non-constant object. So, I have a private member, I have a public member, please note that here I have not written private, if at the beginning, I do not write anything, then it means private always. And then I have written so I have a private and public member, and I have provided get member and set member on my private member vs.

Public, I do not need to do I can directly access that. So, I can make all sorts of changes, I can read both of them, set both of them, print and all that. This is the usual non-constant scenario.

(Refer Slide Time: 02:58)



What happens if I make things constants, all that I have changed is in my object creation, declaration time, I have made it constant like const int, I do a mid const MyClass. So, it says that if it is constant, then what is the change that is going to happen? The change will be that this pointer of my constant obj is now cons MyClass because it is a constant object.

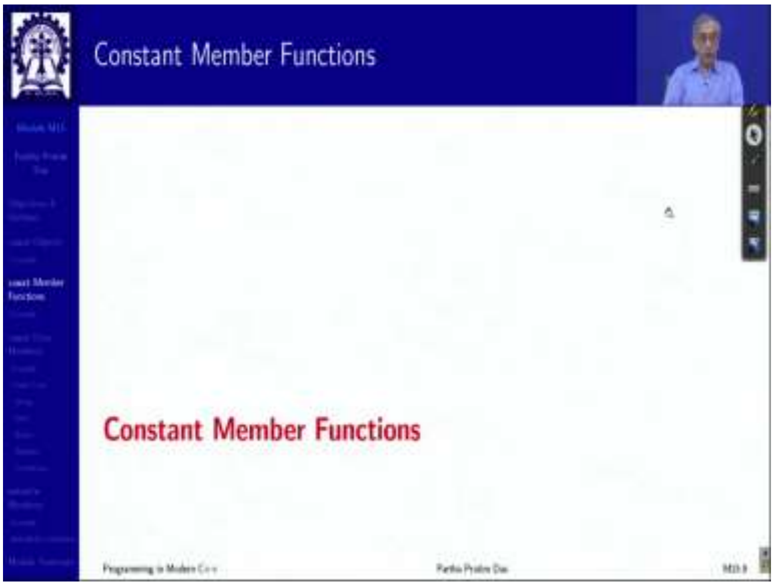
So, it is this pointer has changed with this const whereas, these functions, member functions that you have written whether it is a get member or a set member or print all of these expect a this pointer which does not point to a constant object.

So, even you would see that why is getting member not working is get member is not actually changing it get member is just getting the value, but the compiler will not make it work because get member here expects this pointer of the type which is constant by itself, but does not point to a constant object, whereas what it gets is including this constant, so there is a type mismatch and it will not allow that call to happen.

So, with that, you can you can see that this will not work. Obviously similarly print will not work set member will not work. None of this will work. This will also not work. This is there is no function call this is just an assignment; this will not work because that is your basic state. So, you cannot make an assignment to that.

So, that is the, that is the simple interpretation of the constant object. So the key point to note here is you cannot change anything, that is fine. But the key point is even if you want to invoke a function, which does not change anything, you are not being allowed to do that, because the compiler cannot distinguish.

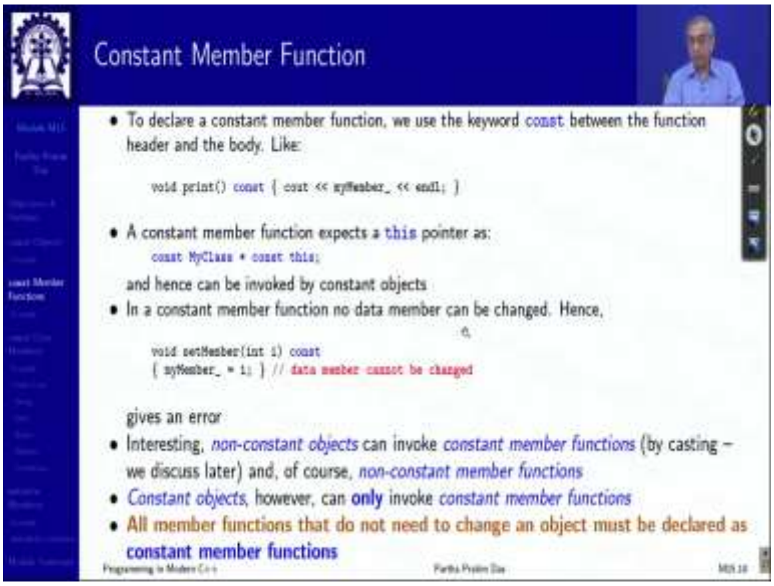
(Refer Slide Time: 5:32)



So, we move on to handle this situation that in a constant object, I should be able to invoke member functions, which does not change the object with the constant objects in the state cannot be changed.

So, if I have a member function, which does not change the state of the object, it should be okay to call that, if it had if there is one, which can potentially change the state of the object that must be guarded that must be because otherwise the const-ness will disappear.

(Refer Slide Time: 6:11)



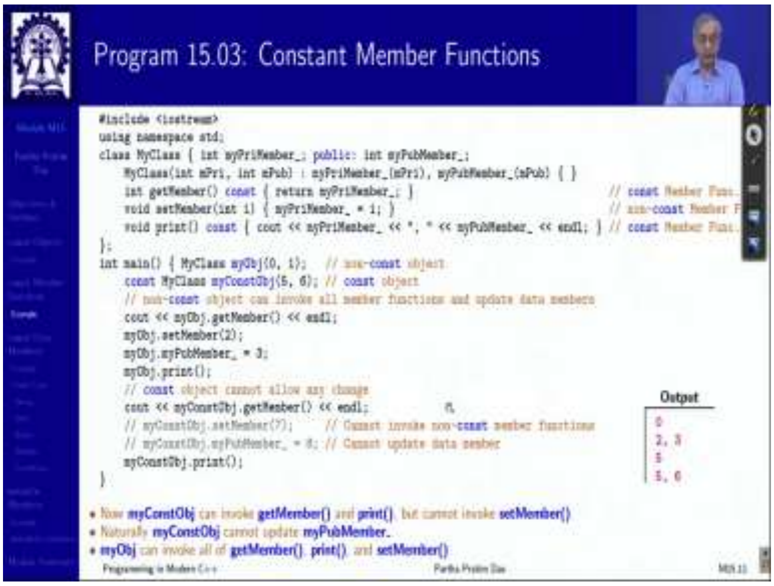
So, C++ introduces what is known as constant member function, what is simply does is it puts the keyword const after the prototype header, and before the body between these 2, the header and the body, it writes constant what change does it bring into the compiler, it tells the compiler that pass a const MyClass * const this pointer as a this pointer to this function.

So, now, the match will happen. One, the secondary tells the compiler is that this function is a constant function constant member function. So, do not allow any change to happen to the state of the object within the body of this function.

So, if I, if we look at 2 functions, there is print, which just accesses the member reading the value, and there is set member, which I have also defined as const, where I am trying to set a value now, this will be fine, but this will be a compilation error, because in a constant function, you cannot change the state of the object.

So, now, you have the full story covered constant object does not change states member functions, anything that I want to use without doing a change, I can define them as constant and anything that changes the object cannot be invoked on this particular constant object. So, if I define a function to be constant, then try to do something within that to change the state, it will not be allowed in the compilation. So, that is the basic idea.

(Refer Slide Time: 8:12)



So, now, let us look at this. So, I again have 2 members, public and private, the constructor the get member I define is const. That is even if the object is constant, I must be able to get that there is no problem. The set member is done as non const then that is it will not it will take myClass * const this because it wants to change the data member that clearly tells the compiler this function will change the state this function will not change the state.

Similarly, print I make const because I only want to read the public and private members and print them I do not want to change anything. Now, let us define 2 objects one is myObj and another one is myConsObj, this is non constant, the second object is constant. Now, if the object is non constant, then I should be able to invoke any member function.

If the object is non constant, I should be able to invoke anything it does not matter. So, I can invoke the non-constant member function, I can also invoke the constant member function because the constant member function says that I am assuming the object cannot be changed. So, if I give it an object, which is okay to be changed, and that function does not change anything, I am not violating anything.

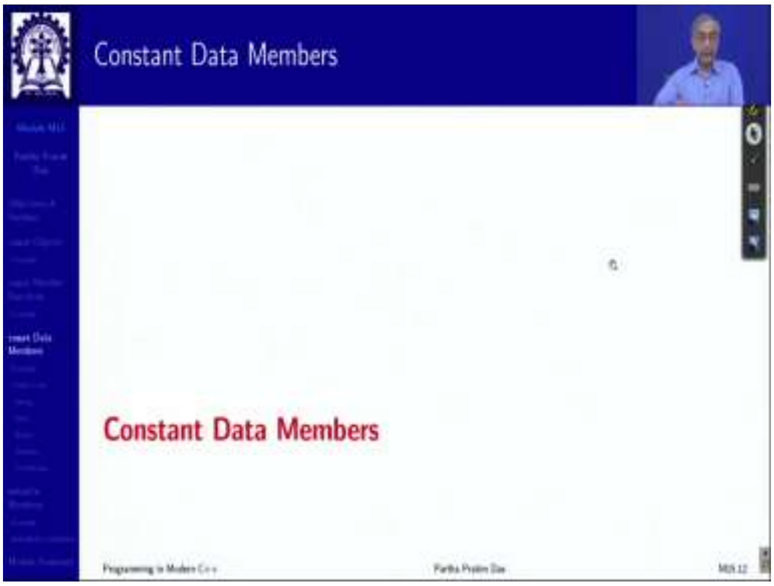
That is in other words, you are in other words, any pointer like X * const this and const X * const this where X is the name of the class, any pointer which is non const pointing an I mean any pointer which is pointing to a non const object can always be treated as if pointing to a constant object there is no violation. So, this is what as we will see is a valid pointer cast.

Whereas, if I try to do this then it is a violation if it is this is saying that the this pointer is pointing to an object which is constant I cannot change that to a pointer which is pointing to a non-constant object. So, for that, so, the consequence as you will see is that on the non-constant object I can do anything of course, I can do direct assignment also.

But for the constant object, I can do this which is a constant function get member, I can do this which is a constant function because the guarantee that they will not make change, but I cannot invoke this if I invoke this, I am trying to change a this pointer pointing to a constant object with this pointer, which is pointing to a non-constant object which is a violation.

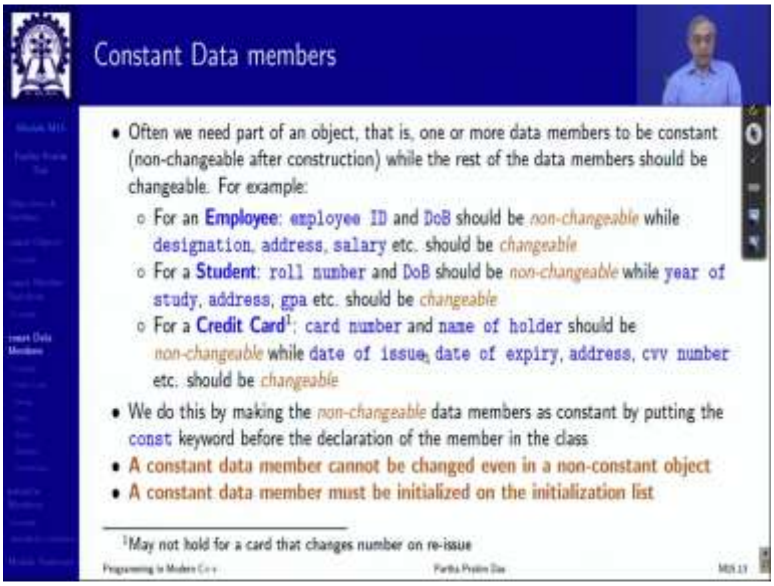
So, if I try to do this, the compiler will not allow me it will give me a compilation error and obviously, I cannot do the assignment.

(Refer Slide Time: 11:39)



So, by combining the const-ness of the data member and the cons and const-ness of the object and the const-ness of the member function, we can easily invoke any member function.

(Refer Slide Time: 11:49)

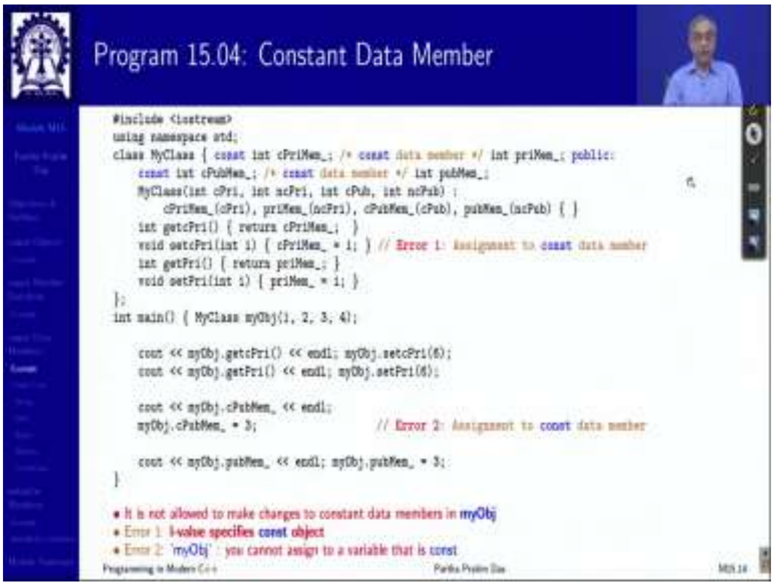


Now, let us see what happens with the data members. Now, if I make a object constant, then the entire of the object is constant. But oftentimes, I have designs of classes where only a part is constant. For example, the several examples can be constructed here is an employee employee's ID is not expected to change employees Date of Birth certainly cannot change.

So, once you have created they are fixed, whereas, the employees designation employees address salary this could change for a student the roll number and date of birth will not change, but suddenly address by GPA year of study will keep on changing for a credit card, maybe the credit card number and name of the holder should not change but others can change.

So, it is not enough to be able to just define an entire object as constant, I want to make just specific data members to be non-changeable as well and it is just an extension of the idea of const-ness which let me do that.

(Refer Slide Time: 12:53)

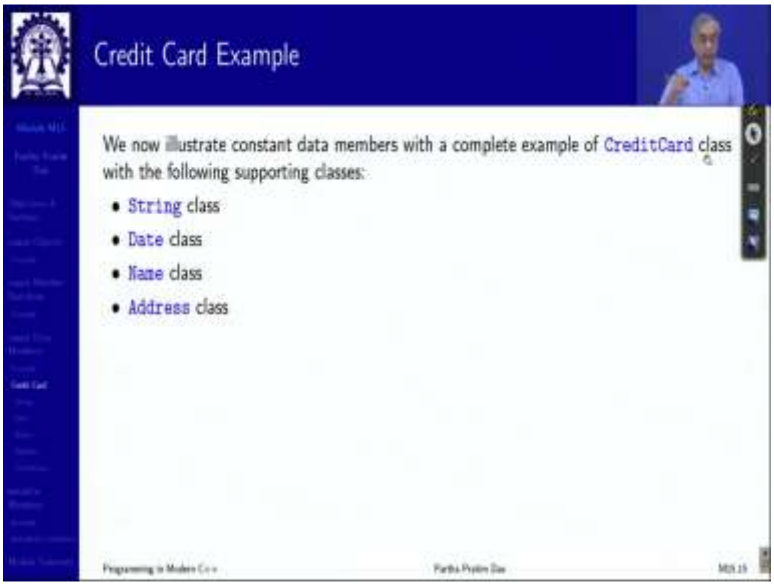


So, now I have 4 data members created 4 data members in this design for our understanding. So, there is a data member cPriMem_ which is a private constant data member, which means that I will I should not be able to change it ever, whether the object is constant or the object is not constant, and then I have a normal private member.

Similarly, I have a public constant data member I have a public normal member I have a get and set on the constant member. Now, the gait obviously, will work because I can read, but the set is trying to change this mind you the object is not constant, the object is non constant object is non constant, but I have told that this particular data member must be constant.

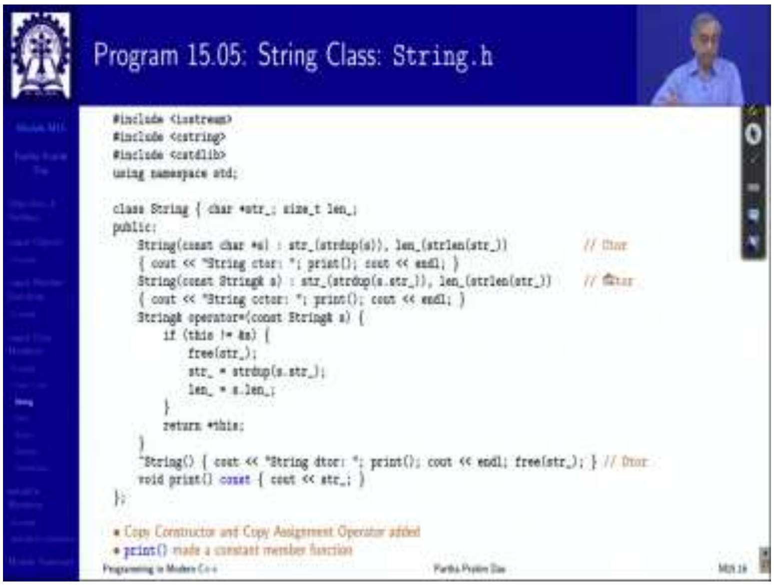
Therefore, making any change to that is not allowed and the compiler will give you this assignment error whereas, if you want to try the other one this one which is a non-constant data member, then you will be able to read it as well as write it. So, if you try to try this out, then you will see that there is a problem with the setcPrime member function which cannot be compiled because of the const-ness is similarly here you cannot make a direct change to this data member even though the object actually is non constant.

(Refer Slide Time: 14:44)



So, we using this you can you did a practice I hope of the credit card and related classes earlier. So, now you can introduce the const-ness in this very nicely and make a design, which can keep all these promises of what values should change and what values should not change, I am sorry.

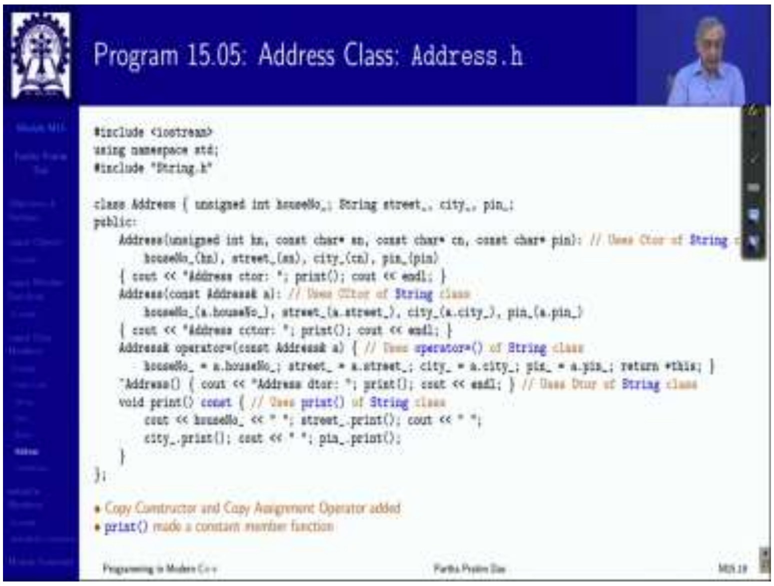
(Refer Slide Time: 15:10)



So, this is a string, this is a string class, which is just for support, this is a date class that you have seen, what I have added, we have added const here. That is it is at even if the date is constant, like in date of birth, I should be able to print it, I should be able to validate it. And I should be able to say which day is it, right, so there is that. So, and we have also included the copy constructor and copy assignment operator to be able to work perfectly.

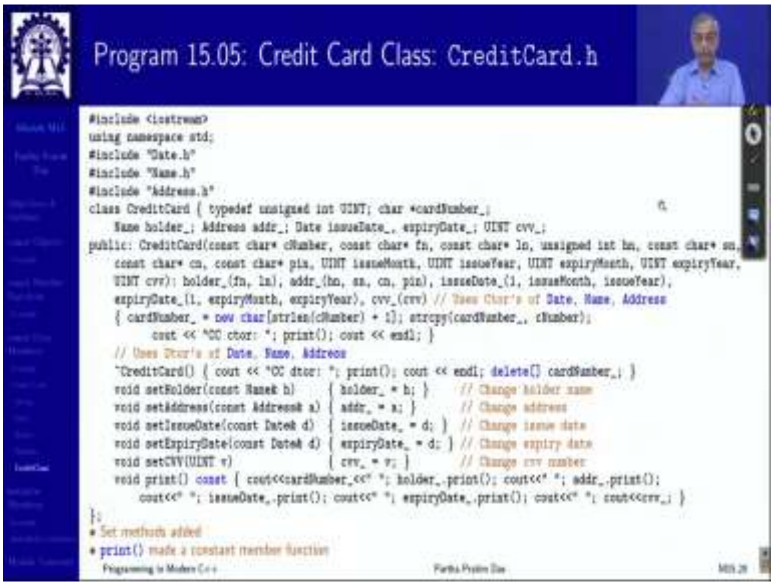
There is a Name class, in the Name class, we have the print made into a constant member function so that I can print anything that I want. Additionally, we have put the copy functions, we are now just trying to show you how slowly the design builds up.

(Refer Slide Time: 16:22)



Then have the address class. In the address class, again, the print is a const function, copy and assignment operators are added. I am just flipping through, you have to really study this code, understand how to write them again and try it out on your compiler and see what you are getting.

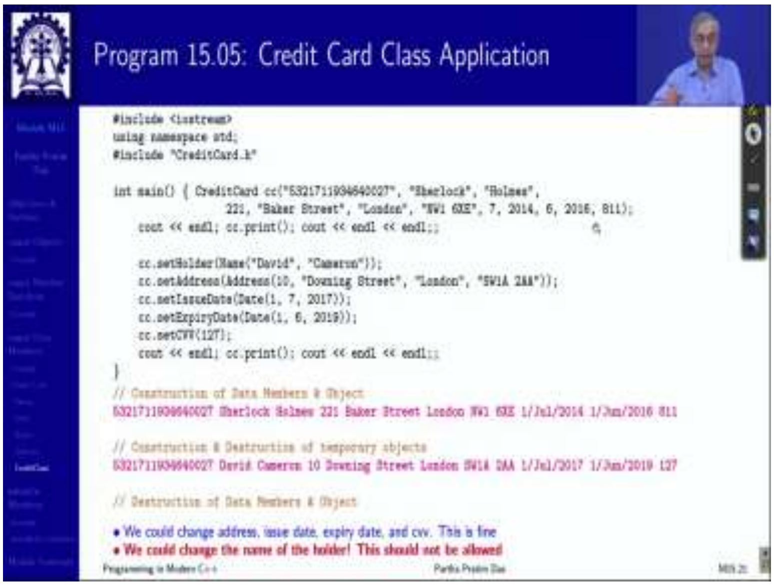
(Refer Slide Time: 16:43)



So, finally, the credit card class the Credit Card class has again, the same type of changes print is made const you have, you have the total credit card constructor. And you have been it will be able to with this construct any credit card information.

So, you have credit card constructor here, you have the red card destructor here and you have after construction, you have all different functions given to be able to change the respective values like the name of the holder, address of the holder, issue, date, expiry date, CVV number and so on so forth. That is our initial design.

(Refer Slide Time: 17:36)



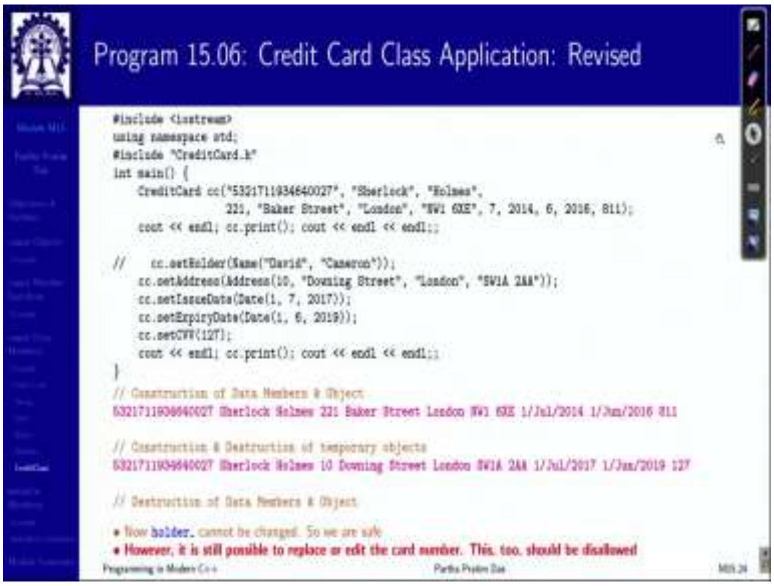
Now, I write an application with that. So, this is the object I am constructing the card name, the holders name Sherlock Holmes, the holders address 221 B, Baker Street in London, and so on, and the date of issue and all those I mean, these are all obviously, you know, meaningless data.

And I can change that to the name of say Mr. David Cameron residing in 10 Downing Street, in the data and all that I can make all these changes, and you can see the effect. So, you can see that the designer have provided does everything. But it makes the class vulnerable because I was able to change even the name of the holder.

I have not changed the credit card number. But the name of the holder has been changed from Sherlock Holmes to David Cameron, this obviously is not acceptable, right, it has to be stopped it should not be possible to do this. So, now I introduced the const-ness here.

So, what I do I make name a constant data member that is you can once you have created you cannot change that anymore. Once I do this, then the set holder function will no more compile because I am assigning to that right. So, I have to get rid of this function because if the name is constant, there is no need for it set holder. So, the cleaner one, I have moved the set older function, done clean code.

(Refer Slide Time: 19:26)

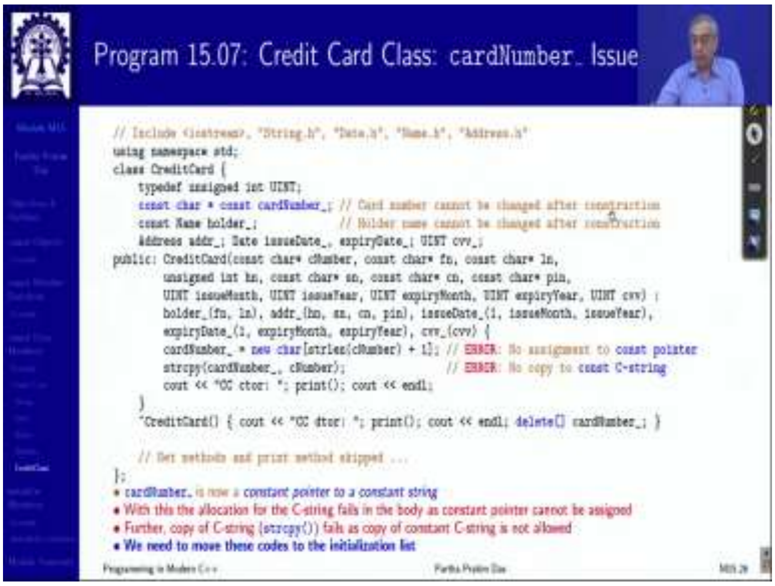


Now, so now I have the revised application. In the revised application, what I am doing is I have removed the set holder function. So, keeping the card number same and just changing the name to Mr. David Cameron is not possible. So, that part of the code I have commented out but it is still possible to edit or replace the card number which should also be disallowed. So, how do I put that constraint? So, I said this is the card number issue.

So, what is the card number card number is a string which is you know dynamically allocated and initialized with value at the at the time of construction, right. Now, to make I want 2 constraints 1 is I want to make the card number non replaceable, that is I should not be able to change that card number and put a different card number. What does that mean? This a pointer. So, I need it to be a constant pointer because then I will not be able to change that card number to something else you have already understood.

The other is I also need the card number to be non-editable, I should not be able to change few digits in it which means that card number pointer, the string that it is pointing to that should also be a constant it should be a constant string. So, requirements of having the card number non-editable and non-replaceable is to convert this pointer into a constant pointer to a constant char*.

(Refer Slide Time: 21:28)



So, now, I have done that have made it into constant pointer to a constant char*. So, it is it cannot be changed after the construction, the name of the holder also is constant. So, that cannot be changed. So, with this now, we will have some problem, because when we were doing this allocation in the constructor, this is what we wrote that is at that time, it was not card number was not no const-ness anything.

So, what I said is it is the initialization list has happened up to this point, the body has started body ends here within that I have taken the cNumber parameter of the constructor found out the length incremented by 1 for the null character I have allocated dynamically a character array of that size.

And then once the allocation is done, I have done a string copy. Now, here, the point to notice once I have made the type constant pointer pointing to a constant object, the first thing I cannot do is make this assignment because the card member will be initialized with nothing that is garbage in the initialization. Once I have entered the constructor body remind you in terms of the object lifetime has started.

So, now I have an object. So, card member is now an object by card member is now a data member of an object which data member must be constant because the object lifetime has started. So, if I had to do anything with this pointer, I had to do it in the initialization list, I cannot do it here. So, this will have an error because I am assigning to a constant pointer. In the other also when I am doing and strcpy.

This is quite obvious because I am trying to copy and change the string that is pointed to by the card number by trying to make a copy which obviously is not allowed. So both of these will fail. So, we have a very nice design in terms of the data member of the card number pro having protected it for making it non replaceable and non-editable, but I am not able to construct object.

So, what I have to do is you must have understood that by now is I will have to use all this in the initialization list itself. Because once I am here, no changes can be made the object lifetime has started. So, let us see how to do this final correction.

(Refer Slide Time: 25:00)

So, so I say that card number issue dissolved. So, this is what we have. Now what I do, I have put a single line initialization for the card number, just see how I am doing it, because you will often need to do this thing, particularly for constant string to constant data, I have taken the found out the length how much of a big size array we need, I allocate that and use that as a destination for strcpy.

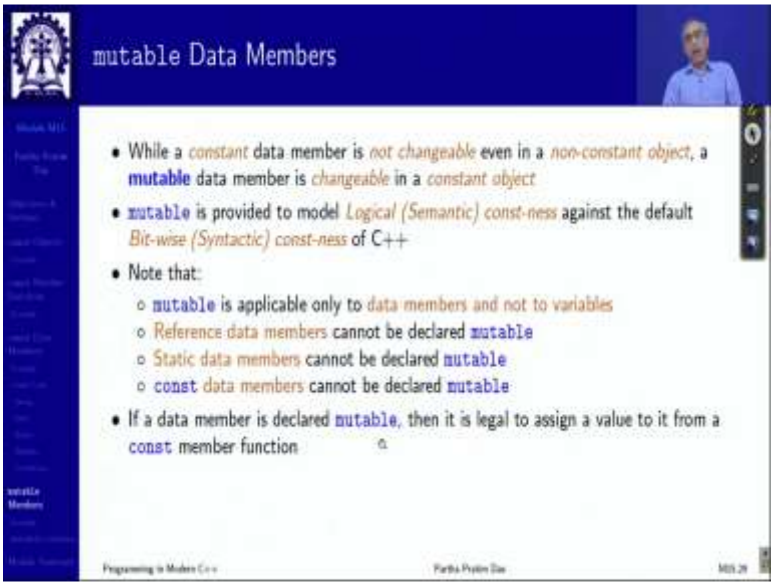
And the source is C number, the beauty of strcpy is it returns the copied string, same pointer, right. So, whatever has been allocated here, will be first used to copy cNumber into that, and that that same value will be returned by strcpy.

And get initialized in the cart number. So, this is a very typical code. And you might just, you know, want to try it out with different types and get convinced but this is a very nice way of making sure that if I have a constant pointer to a constant string or constant object, then how we can initialize it and get started.

So, once I have that, naturally, the body has a constructor has no further code other than just printing the object just to make sure that you know, we get to see what is happening right. So this is my card number completely unchangeable.

This is my holder name, which cannot be edited. I am ready with the kind of design I wanted for my credit class, credit card class and all data are initialized in the initializer list. And that is preferably should be done. Unless there is a very compelling reason to put some initialization code in the constructor body. If you design it, well, it will not be there.

(Refer Slide Time: 27:18)



Before we conclude, I would like to just remind you of another feature what you have seen is constant is all over the whole object can be constant, and for that to deal with it I may have constant or non-constant member functions data members can be constant. But what is this const-ness that means a constant data member means that it cannot be changed even in a non-constant object.

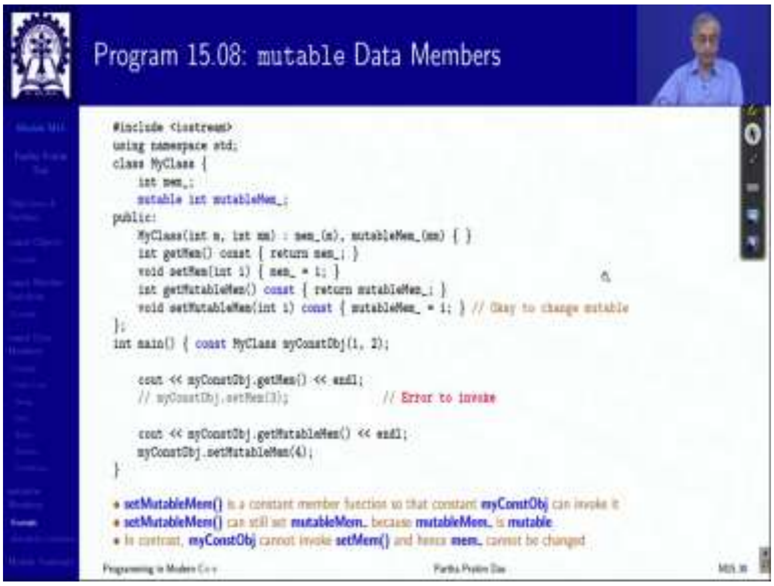
So, either the whole object is constant or a data member is always constant. But if I want to say that well, I want to change a particular data member even if in a constant object. How do I do that? So, that is where the whole concept of mutability and mutable you understand something which you can change.

So, the const-ness as we have treated so far, is again called the bitwise const-ness that any data member is const-ness, you cannot change any between. Remember, a whole object is const. You cannot change anything in the state.

But that bitwise syntactic const-ness is not always enough, I may want logical or semantic const-ness, which is what is provided by mutable so I can say a data member is mutable so that even when you consider the whole object constant bitwise that particular data member, I would be able to change. That is a specific semantic context in which it comes in. And it is applicable only to data members.

There is nothing like a mutable variable reference data members cannot be mutable static data members, we have not discussed yet when you come to it cannot be mutable constant data members obviously cannot be mutable. So, if a data member is declared mutable, then it is legal to assign a value to it from a constant member function, which is not possible for other cases.

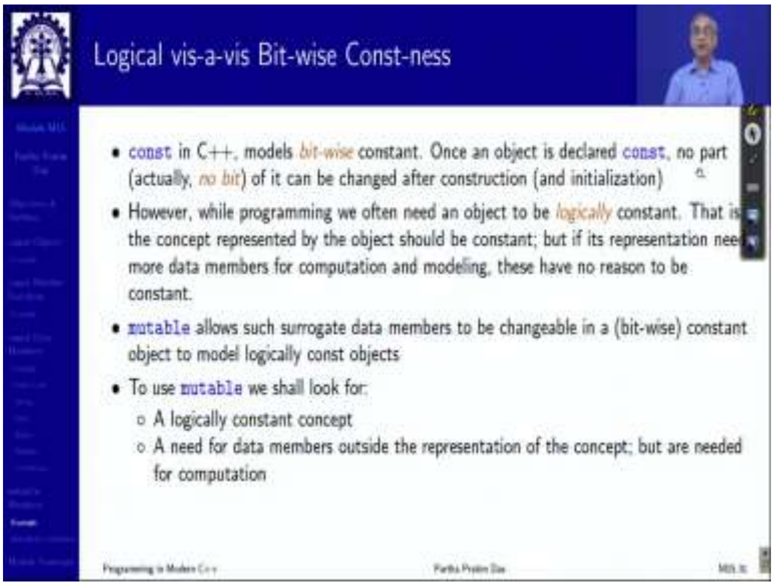
(Refer Slide Time: 29:30)



So here, I have a mutable, I have a normal member and a mutable member a constant member function to read that is, that is fine. But what I have is a constant member function to set and I can set the mutable this member here. I will not be able to set mem_ here, because not mutable. So, once the function is constant, once object is constant, it cannot be changed. But this one, particularly just this part can be changed.

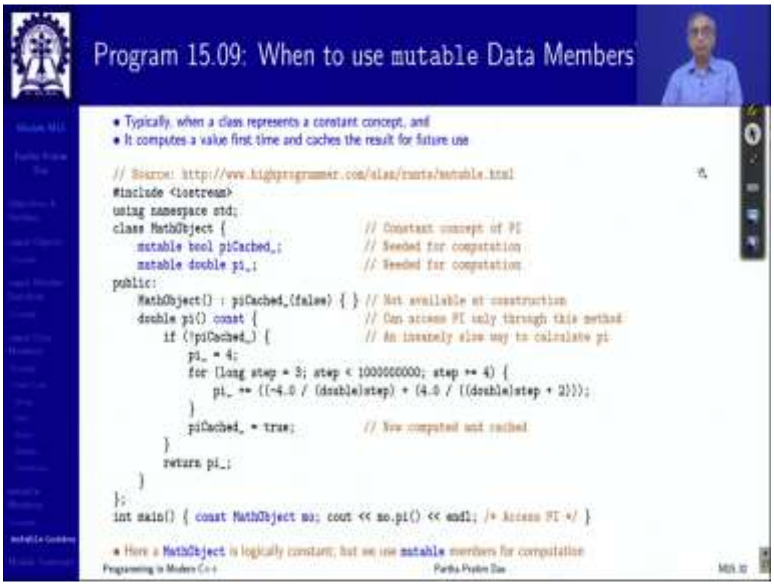
So, if I define a constant object, I can invoke get mem on that, because it is constant, I cannot invoke set mem because it is not a constant member function. This we have already seen, I will be able to invoke get mutable mem, because it is const, I will also be able to invoke set mutable name, because it is const, though, it actually changes the value of the variable. So, there is a very fine grained contextual control on the const-ness that is provided.

(Refer Slide Time: 30:46)



So, const in general in C++ is bitwise constant. So, once the object is declared as constant, no part no bit can be changed. But to support const-ness, which is logical, which often programmers need the mutable feature is provided. So, to use, mutable we will look for a logically constant concept, and data members need for data members outside the representation of the concept but needed for computation. quick examples.

(Refer Slide Time: 31:28)



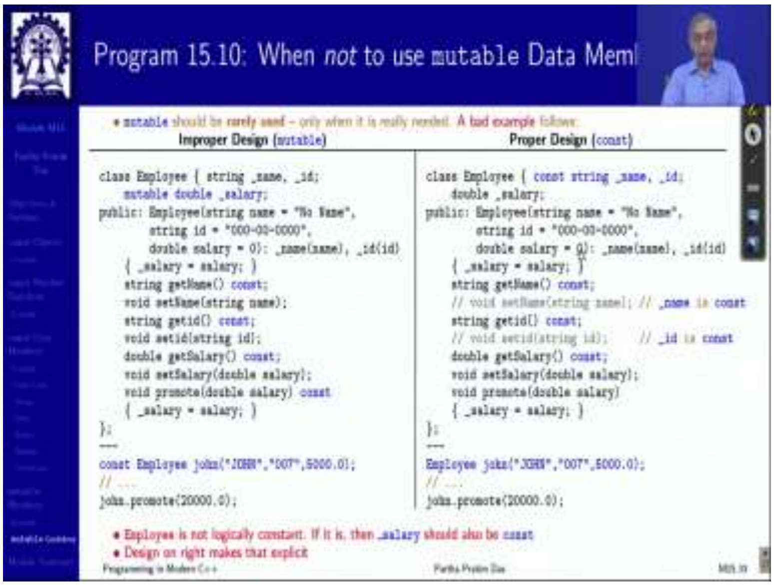
So, I will just show you positive as well as negative example. So, this is when to use mutable data members, for example, here, I am showing that I have a math object, which has a function pi. So, on the math object, I call pi, I will get the value of pi. So, to compute the pi I have provided and what this algorithm is terribly slow and all that, but that does not really matter. What I do is I cache this value of pi.

So, I have a cashable Boolean, so, what it does the first time at construction, it is put false. So, the first time I call pi, it is false. So, it does this slow algorithm, but then the value of pi will not change to whatever value of pi has finally been computed, after this long process. It will be remembered in pi. And I say that the cache is true. So, from the next time, it will just look up that well, fill it in pi.

So, that is it, that is the that is a typical design. Now, in the in my use, I have obviously my object has to be constant, because the value of pi and so on cannot change. And I am invoking this which is a constant member function, it has to be because otherwise on constant object I cannot, but I needed to change the both this I needed to change the value of pi in the context of doing it first time, which is a computational requirement, logically pi is constant.

But the first time I am doing the pi is not there. So, I have to be able to compute and put it so that is the reason I am putting this mutable and to support that pi cache is also mutable. So, even in a constant object, the variable the data member is made mutable so that I could compute it on demand and henceforth, whenever I want I will use it naturally these are these are somewhat advanced risky design concepts. So, you have to use them properly.

(Refer Slide Time: 33:43)



The second I show is when not to use mutable, you know suppose you have an employ class and you say that mutable double salary because the name cannot be changed, it cannot be changed. So, it is fine you say that okay. I will make the entire employee object constant and allow the salary to be changed with the salary can change.

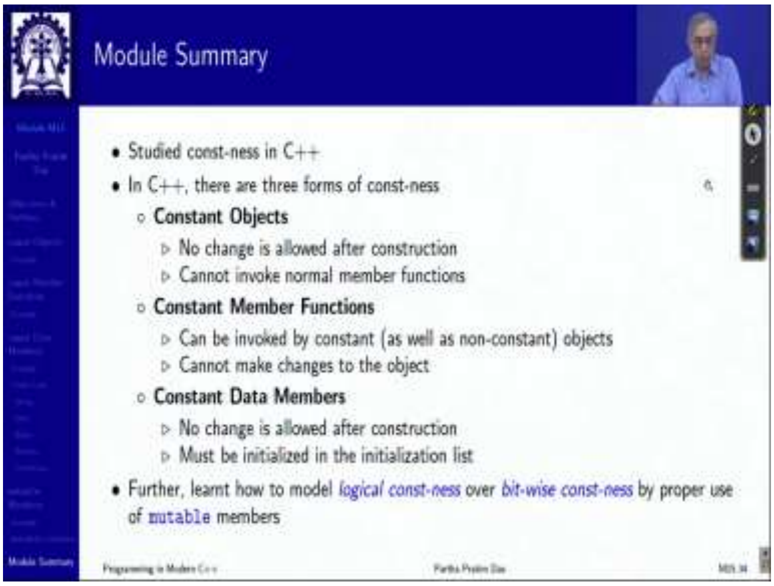
So, I have getName, getId, getSalary all this as constant member function, I have promotion as constant member function and so on. And it changes a salary which is allowed because it is mutable. This is a bad design. programmatically, yeah, you will get through, but this is not the design for which you should use mutable because conceptually employ is not a constant.

Conceptually an employ object is not a constant only its employ’s name and id are constant but otherwise it is the it is an evolving object. So, this is this is not a preferred design. What you should do is rather make name and id constant and there is no need to make it mutable because you should not be treating employ objects as constant.

You just make sure that name and id cannot be changed so, get name will still have to be constant getId will have still have to be constant, setName and setId must not be there because you will not be able to change them, they are a constant, getSalary will be a constant and promotion would be there as a normal member of promotion.

So, please understand this difference that it is not that you can just use mutable with cons and make something work you have to ask yourself conceptually is this object constant. So, by conceptually whatever is constant you go by that, then rest of it you handle either by const-ness of data members or by mutability of data members and you support your design through the constant member functions.

(Refer Slide Time: 36:08)



This is in just what your you know design of const-ness should be doing talked a length about various kinds of const-ness and mutability, Thank you very much for your attention and we will meet in the next module next week.