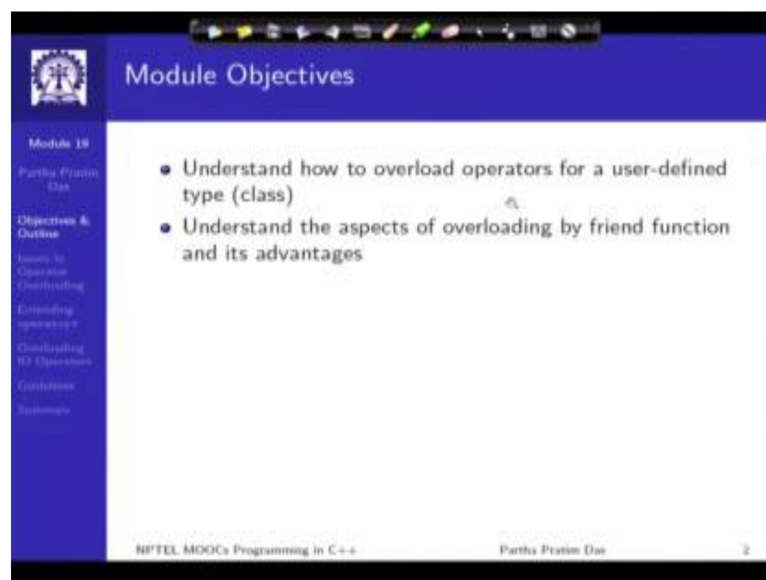
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 34
Overloading Operator for User - Defined Types: Part II

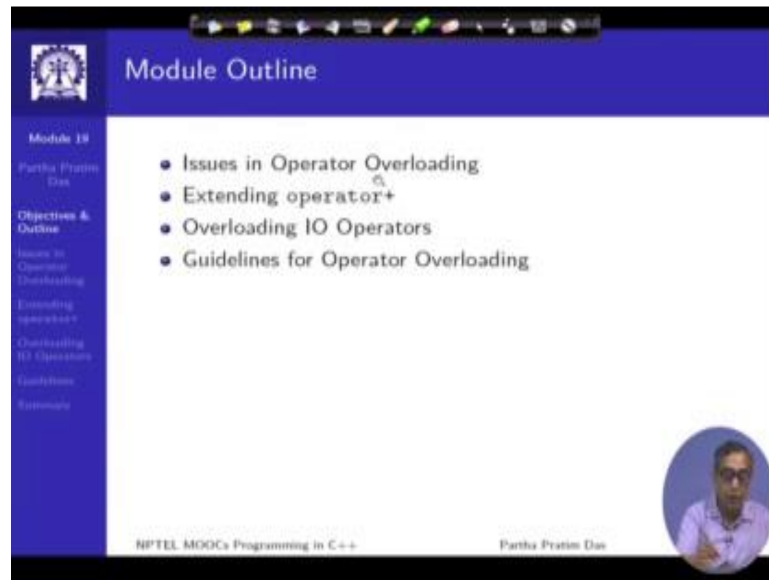
Welcome to module 19 of Programming in C++. All we have been discussing from the last module about Operator Overloading for User Defined Types.

(Refer Slide Time: 00:34)



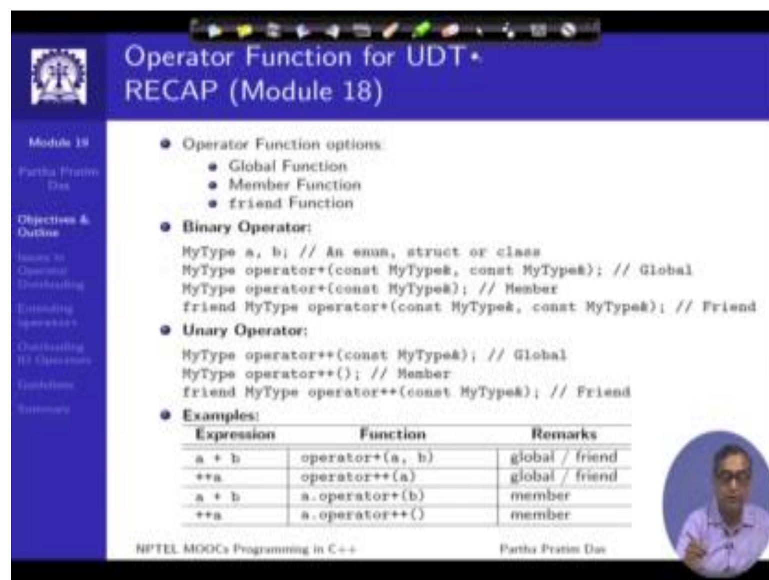
We have seen why operator overloading is important to create a specific complete types with possibilities of writing expression and creating algebra of different types. We have looked at how to overload operators using global functions and member functions. In this module we will take that forward, we will see that even with what we have proposed in terms of operator overloading using global and global functions and member functions there are some issues that crop up in operator overloading and we will show how friend function can be used for overloading operators properly.

(Refer Slide Time: 01:15)



So, we will talk about issues in operator over loading and then specifically, we will discuss about extending operator plus and about the IO operators these outline will be available as you know on the left hand side of your slides.

(Refer Slide Time: 01:32)



Just a quick a recap from a the earlier module, there are three ways to overload operators,

these are the different operator functions that could not have for binary, these are the operator functions for unary and these are the different ways the invocation will actually happen based on an expression that we write this is how the functions will get invoked. We have already seen this.

(Refer Slide Time: 02:01)

Now, let us talk about a extending or a designing. Let us go back to the consideration of the complex class. We have learnt how to overload operator plus to add two complex numbers so we can do this, d 1 and d 2 are two complex numbers we can add them we have seen how to write this using global function as well as using number function. Now what you want to do? We want to extend the power of this operator we are going to say that at times I want to add a real number with a complex number. A real number can be thought of as a complex number with a zero imaginary unary part.

So, could I extend this operator so that I can also write expressions like this, where d 1 is a complex value, where as this is a real value I would what to like to add that or, in a committed form there is a real value 4.2 and I want to add a complex number d 2 to that. Will my operator plus design be able to take care of this? If it can, then I really have a good value for my addition operator here. What we will try to show that what are the difficulties of doing this extension with the global function and member function and

how does a friend function help in this solution.

(Refer Slide Time: 03:20)

Issue 2: Overloading IO Operators: operator<<, operator>>

Module 19

- Consider a Complex class. Suppose we want to overload the streaming operators for this class so that we can write the following code:

```
Complex d;

cin >> d;

cout << d;
```

- Let us note that these operators deal with stream types defined in iostream, ostream, and istream:
  - cout is an ostream object
  - cin is an istream object
- We show why global operator function is not good for this
- We show why member operator function cannot do this
- We show how friend function achieves this

NPTEL MOOCs Programming in C++ Partha Pratim Das

Then we will also take up a second issue that is of overloading IO operators. Again consider the complex class and we know for example if I write in parallely in case of say a double type, then if I have a variable of type double then I can input from c in I can output to c out using these operators. Our intention is for the complex type also I should be able to do a similar kind of expression I should able to write a similar kind of expression and functionality and that is basically will mean that the operators that we have the input operator and the output operator I should be able to overload them.

In this connection it will be good to note that whenever I do the IO if I am using c in, c in is actually an I stream object input stream object, whenever I do c out that actually is a O stream object or out stream object. We will have to a really design these operators so that we can overload keeping this stream types appropriately in our design. We will again show in this module that why global and member function solution for operator overloading fail in such cases and how the friend could help.

(Refer Slide Time: 04:52)

So, let start by taking to extend operator plus with; first say, let say if I want to extent so by extension what we are trying to do we are trying to, let us looking to this code we are trying to add these two lines. We already have this we know how to add two complex numbers. We want to add a real with a complex. This is we are doing by global function so this overload one is the function that we had seen earlier which takes two complex numbers and add same component wise and returns a new complex number which is a result.

Now, if we want to do this next one which is where the first operand left operand is a complex number and the right operand is a real number then we will design a operator plus which has a pair of a parameters were the first one is complex and second one is double. This 6.2 will go as double, this will go as and that is my overload two, this is just another. Because we can overload as many as many types as we want as long as the overload resolution will get of them of C++ that we had discussed long ago can resolve between these overloads. If this particular version is taken then certainly we expect overload two to take place.

Similarly, if the last one is done were the first parameter the first argument, the left argument is a double then the third overload version should come in to invoke. In this

what we do? We do a sample thing we take this real number and construct a temporary complex number out of that real number which as zero imaginary part and you could just refer to the complex constructor you will able to see how this is happening. Then we simply use the operator that we have, now we have two complex numbers so we just use the operator that we have already implemented in overload one to get them added has simple as that. So operator overload version two and three basically make sure that from the double argument we appropriately create a complex argument and then call the overload one. With this certainly the problem gets solved three separate overloading solves the whole problem, but we know that since we are using global function we have all the new answers of breaking the encapsulation and that is not a very good preposition.

So, next what we will do we will try to use a member functions.

(Refer Slide Time: 07:44)

Let us a move the encapsulation back, now these all have become private. Now we have moved to the operator overload functions here, overload one is here, exactly like the same way the only thing that has become different is the fact that it now has only the right hand side operand as argument to this function and as a parameter to this function left hand side operand is a object itself. If we talk about this particular case which adds two complex number we had seen this earlier it will invoke this overload one. Similarly,

in this case where the argument is a single double number because the complex number on the left hand side operand is an object itself, so in this case it will invoke this overload two, this is fine.

The issue starts with this third form. In the third form you can see there are two issue; one is certainly there are two operands and one of the them is a object and the other one is a parameter to the member function to which you are making the invocation. So the consequent issue is, if I have to do that for 4.2 plus d 2 then the member function has to belong to the class of 4.2, because that is the left hand side operand where the invocation will happen. Now 4.2 is of double type which are built-in type and naturally I have no option of adding another operator to that or overloading the operators in the built-in type.

So there is no way that I can actually write for this, either expression we will not be able to support if I am using member functions for overloading my operators so that is the limitation of using the overloading with member functions because only some of the intended semantics overload 1 and 2 is what we have been able to support but the overload 3 will still fail, because we cannot write in a appropriate function for that.

(Refer Slide Time: 10:04)

So that is basic difficulty of what we are saying as issue one. So the solution is very

simple and elegant we have already discussed about friend function in depth. So all that we do is we go back to the solution of the global function which was the perfect solution, but the only difference being that the needed to expose the encapsulation. We know that the front functions can actually look inside the class, so what we do? We go back to that solution and basically make the data members private as we would want. But move all the global functions into the scope of friend function in the class.

The same signature has the global function with prefix to with friend and put inside the scope of the class and then we implement them. They just behave like the global functions only, but being friend they can access directly the private data members, so here, if we now have this particular function.

Now this the overload two is called where the second argument is a double and the overload three is also possible where the first argument is the double and depending on that, it gets resolved and now since it is a friend function it can access the internals of the complex class and actually it is possible to implement them even though the parameters re and I m are private members. So friend gives us a very elegant solution for the whole a requirement and we will be able to achieve exactly what we needed without breaking the encapsulation that we were requiring earlier.

Just in the passing I would like to make a pointer to a note that I have put in here, is you will see that in this many of this solutions we have actually created three overloads which are very similar and related and these two overloads are primarily to convert the double in terms of the corresponding complex number and C++ does have a feature now known as the implicit casting which can do this task without explicitly being written by the user.

We have not yet studied about casting so when we studied that we will see that we will have even more compact solution where only overload one function would be able do all the three tasks without any difficulty. Right now I do not want that casting to happen and therefore, I have used a keyword explicit in front of the constructor which basically tell this compiler that do not do any casting for this object. So we will talk about this when we talk about the casting later on.

(Refer Slide Time: 12:58)

Let us move and talk about the second issue the IO Operators. Let us first try to understand issue what are we trying to do, there are two complex objects and I want to write this, this is the basic thing, I want to use this and be able to write them. Let us understand that the order in which these operators are applied are from left to right, so basically when I write this it means this. That is this operator, output operator is a binary one and it takes two operands; if you look into this the left hand side operand is a c out object and the right hand side operand is a d 1 object. So I can say that in my signature how should this operator function look like the first operand is of the O stream type because c out is an object of the O stream type.

The second operand here is of the complex type because d 1 is an object of the complex type. You will always have to work with signatures which match the desired use of the operator. Now you are left with the question of what should be the return type, what should it return, should it just return a void? No, if it returns a void then the difficulty is we will not be able write this. Because if we change the output operator then it means that we first output this then you output this. Now, you think about this instance of the output operator. This is the right hand side operand and this whole parenthesize things is the left hand side operand. So if this has to be there left hand side operand, but this is the result of a computation of the earlier invocation of the operator.

So, that result must be an output stream operator. So if I need to change and I would refer you to a recall the way we had discussed about operator assignment we had similar discussion there as well that the return type of the this operator is decided by what is your operand in the input, because whatever you written here should be able to go back as input to the next invocation of the same operator. So that justifies that this is the signature that I need to ensure if I am implementing this operator as a global function. Of course, I can implement it as a operator as a member function. Now if I implement it as a member function naturally since there are two classes involved here O stream and complex I have two choices for member functions. One where it is a member in the O stream class and it takes complex as an input, or it is a member in the complex class and it takes O stream as an input. So, given that a let us try to see how the operator will get deigned.

(Refer Slide Time: 16:09)

Let us try to implement it with the overloading the IO operator using as a global function. So this is the basic overloading. The signature we have already discussed so what we need to do in terms of the actual output we will just need to take this two components and print them with a plus j for the complex number notation and that should achieve my output operation and since I have to written the same output stream operator so whatever I get in as a parameter here must be returned as a value of this

operator computation. In terms of this I want to use the complex number for printing certainly we could have used a call by value here, but that would mean an unnecessary copy so we are using a reference parameter and as it is a convention we do not expect the output operator to change the value therefore it is a constant reference parameter.

Similarly, I can write I stream operator that is input streaming operator everything look very similar to that only difference being now the complex number that comes in as a parameter has to be a non constant one, because I am doing an input. So with the input I expect that complex number to change. I do the component wise input and the remaining discussion remain the same that.

Finally, I return the input stream that I have got as a first parameter. If you do this then we pretty much have input output for the complex number and we are able to write this and it will work exactly as it does for any int or double types. So the only difficulty in this solution is it breaks the encapsulation, as we pretty much know by now that any overloading with global function will break the encapsulation.

(Refer Slide Time: 18:06)

Overloading IO Operators with Member Function

Module 19

- Case 1: operator<< is a member in ostream class:

```
ostream& ostream::operator<< (const Complex &a);
```

This is not possible as ostream is a class in C++ standard library and we are not allowed to edit it to include the above signature

- Case 2: operator<< is a member in Complex class:

```
ostream& Complex::operator<< (ostream& os);
```

In this case, the invocation of streaming will change to:

```
d << cout; // Left operand is the invoking object
```

This certainly spoils the natural syntax

- IO operators cannot be overloaded by member functions
- Let us try to use friend function

NPTEL MOOCs Programming in C++ Partha Pratim Das

So let us take the same step let us try to do this by member function. Now if we want to do this by member function certainly we have noted there are two possibilities that the

case one is, operator output say is a member of O stream class. So we will have a signature like this. Now this is not possible. Why is this not possible? Because O stream is a not a class that your i have written, O stream is a class which is provided as a part of the C++ standard library and we are not allow to add members there, we are not allowed to edit that class in any way. So if we cannot edit that we cannot add a new member function like this. So this possibility is ruled out.

I can have a second option is, complex is my class so I can certainly try to overload the operator output in terms of a member function in the complex class so then it will look like something like this, because since I am overloading in the complex class my default parameter is a complex object so all that I need to specify is a O stream operator. But this is a sever consequence. The consequence is my earlier order was O stream object and then the complex object this was the left right order, but here since complex is the class on which this is a member the order will become the complex object O stream object because this has now become the right hand of the operand. So which means that the output done with this kind of an operator if I do then it will have to be written as in this notation d redirection c out naught, c out redirection d the way we are used to understanding this operator.

This discussion will tell you that it is actually not possible to overload the IO operator using member function either in the O stream class or in the user defined type.

(Refer Slide Time: 20:13)

So we are left with only one option that is to use the friend function I have not illustrated that here because you all know, all that you will need to do is to take the global function and more than inside the class and prefix them with the friend and that will simply solve the problem. Before I close I would like to leave you with few guidelines for operator overloading, because we have discussed in different context the overloading actions operator overloading by global function, by member function and friend function. If you are overloading with the global function you will do that provided encapsulation is not a concern that you really do not care about encapsulation, and typically that will happen if you are using structures only.

Like the string example, C-string example we had shown where we are using a structure string just to rap the char star point, just to rap the C-string and then having done that we are able to write and overload for operator plus which can add or concatenate two string object or one string object and one char star pointer and so on. When you have no concern for the encapsulation or little concern for the encapsulation use global functions for operator overloading

Similarly, use member function, when it is guaranteed that the left operand is necessarily a class where you can make it a member. So, we have seen situations where it fails, but if

in the complex number design the operator plus if we did not have the requirement of being able to add a real number with a complex value by operator plus then we can pretty much keep the operator plus overloaded as a member function of complex.

But in other cases where the operands might be a multiple types or it is not possible that you can make the appropriate left operand of a type where you can add the member function you will not able to use that. So whenever possible use member functions as for operator overloading. And in all the remaining cases and the major cases we have already seen and that you will find that lot of operators are overloaded in that same style you will have to use the friend function for overloading the operators.

Further to this I would also like to highlight that whenever you overload an operator basically you are writing a function, so you are preserving the arity, the associativity, the precedence, you are basically trying to create a new function and you are associating that function with the operator symbol. Now most of us, the programmers have been custom to certain semantics for the operator symbols, we are used to thinking in a certain way. For example, if I see an operator plus then in some way I get a sense of a adding things putting things together making a union and things like that. If you are using operator plus for a certain type, you should look for does this the semantics that you are putting in the overload of the operator plus does it sound similar to doing some kind of a union addition kind of things. For example, if you are overloading operator plus for a set class then a certainly you should use that to compute union and not to compute intersection.

Those kind of things that a naturality, the more you make the operator overloading for your type closing semantics to the behavior of the operator for the built-in types you will find that your class will behave more nicely and other programmers would be able to understand and debug your class in a much better way there will able to use your class in a much better way.

Similarly, for passing the parameters to your operator function try to follow strict to the default parameter passing the invention which is, as I had mentioned several times is if there is a built-in type you pass it by value and if there is a user defined type then you try to pass it by constant reference. Certainly you will have to be careful about the

requirements for example, we just saw that if we are doing an input streaming operator then we need to pass the parameter the data value as a reference which is non constant because we expect that the input streaming we will actually put value into that, but otherwise first attempt would be to try to honor the parameter passing convention.

In terms of the return type again you try to rely more on the natural semantics of the built-in types for example, we just discus that the input direction and the output direction, input streaming and output streaming operators have been designed with changing so you would preserve that property when you overload the input a streaming and output streaming operators. They should be changeable you should not or write it in a way so that I cannot change the output stream or the input stream. And therefore, that has dictated that the return type of this operator has to be same as the left hand side operand of the operator.

And that is what I mean by the natural semantics of the built-in type or another example of natural semantics is as we discussed in case of pre-increment and post-increment cases. For pre-increment for it is built-in type it is a original object which actually a gets increment and then object comes back so you should use a return by a reference as a refer post increment you get an old value of the object so it will have to be a new object. Therefore, you cannot get it by reference you will have to get it by you will have to return it by value so please be careful about these decisions. Then the next is you will have to consider the effect of casting on the operands, certainly since we have not discussed about casting I am not in a position to elaborate more on this, but while you discuss casting we will refer back on this. But please do remember that the casting makes a lot of impact on the way you overload your operators.

Finally, I will have a strong advice that please, I mean once you get a custom to overloading operators I have often seen particularly the young programmers they try to overload all operators that are available to be overloaded. And makes a type which has 30 overloaded operators, but in practice may be only 5 or 6 of them are used. So, please while you are making the overload make sure that you make a design which is minimal. Minimal design is a basic requirement of any good programming. And you only overload those operators that you really need actually need to build up your type to build up your

total algebra.

(Refer Slide Time: 27:45)

Module Summary

Module 19

Partha Pratim Das

Objectives & Outline

Issues in Operator Overloading

Extending operator+

Overloading IO Operators

Guidelines

Summary

- Several issues operator overloading has been discussed
- Use of friend is illustrated in versatile forms of overloading with examples
- Discussed the overloading IO (streaming) operators
- Guidelines for operator overloading is summarized
- Use operator overloading to build algebra for:
  - Complex numbers
  - Fractions
  - Strings
  - Vector and Matrices
  - Sets
  - and so on ...

NPTEL MOOCs Programming in C++ Partha Pratim Das

So please keep these guidelines in mind and then your design of operators should really turn out to be good. In this a module we have actually continued from our previous module, the module 18 and we have discussed about several issues of operator overloading a specifically the issue of doing operator overloading when your parameter types could be varied and when particularly in terms of IO when your two parameters are of two different class types and so on. And we have shown that using the friend function there are really elegant nice solution to operator overloading, so I will also refer back to our discussion on the friend function module where I had mentioned the three situations under which the friend should be used as a function and this was a third situation where you really need to use friend, and you can now realize has to the importance of friend in doing any nice design of operator overloading.

We have I have also met some guidelines for you, in terms of how you should overload your operators or which operators you should be overloading and so on, please keep those in mind. And finally, I would advice that in module 18 when we started, we started with a motivation of building complete types, complete algebra for variety of different arithmetical types and others, so a complex number, fractions, string, vector matrices and

so on. So I will art that you practice on the same lines, you practice and start building complete type.

For example, a good exercise would be to try to build a complex type in full which supports all operators that say the int type supports, and it may be actually complex will needs some more operators like you need to find the absolute value of a complex number, the norm of a complex number, you need to find the complex conjugate of a complex number. So, you will have to identify proper operators and overload them. And in that process you should be able to have a complete complex type which would behave exactly like your int type and you will able to write expressions of your complex type values and variables as you do in case of int type.