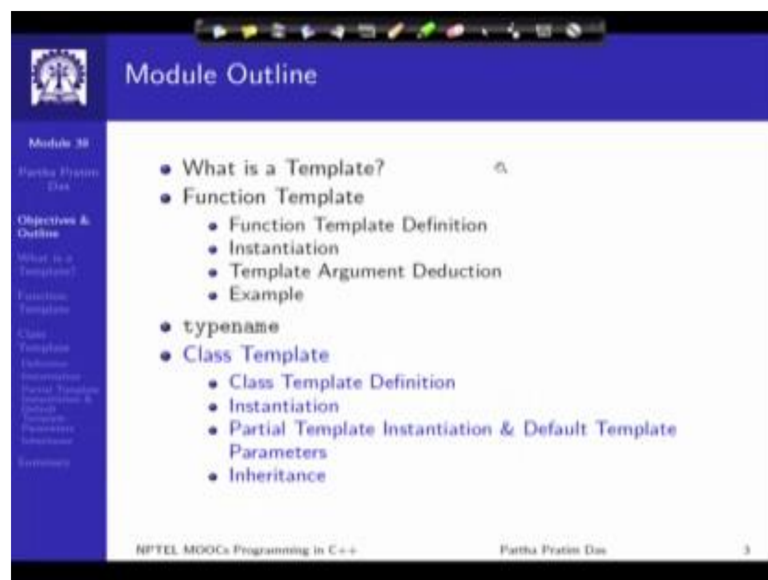
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 55
Template (Function Template): Part II

Welcome to module 39 of Programming in C++. We have been discussing about Templates or Generic Programming, Meta Programming in C++ where we could write some functions templatized functions parameterized functions with one or more type variables so that based on the use either explicitly or implicitly different functions of different parameterized types can get generated as overloads and can get invoked. We have seen a depth in the last module. We have seen at depth an example of a max function which we first wrote in the templatized form used it for int and double and then we specialized it for C strings and then we showed that it will work also for user define types like complex.

(Refer Slide Time: 01:35)



In the current module, we will continue on that and our focus would be the other kinds of templates that C++ has which is known as the Class Template. This is the outline and the blue part is what we discuss will be available on the left of your screen.

(Refer Slide Time: 01:48)

This is just for a quick recap, this is we are seen what is a template and we have seen the function part of it.

(Refer Slide Time: 02:00)

And in terms of the function template we have seen that function templates basically are code reuse in Algorithms. So, you have search algorithm, we have sort algorithm, we

have min algorithm, we have average algorithm and so on. In C++ the code for this is to be return based specifically on the element type. But, in general the algorithm does not change based on the element types so using template we can write function templates which can write this function codes of sorting, searching, min, max, average, all those in a generic form and then instantiate based on that type.

(Refer Slide Time: 02:44)

Now, we can do more if we look into code reuse in terms of data structure. For example, consider a stack, the last in first out. There a several problems which you will stack for example, reversing extreme need a stack of character. Converting and infix expression to postfix requires again a stack of characters. Evaluation of postfix expressions might require integer double complex different kinds of types that we want to evaluate. The depth first traversal of a three would need a stack of node pointer types of the three nodes. There could be several problems which need stacks of various different types to be used for a specific problem solution.

Now, one choice is to write a stack class for each one of this type whenever we need, but what we a looking at can we generically have a stack code which can be instantiated given the particular type that we want because, the stack as a concept is a last in first out with a set of few interfaces like, push, pop, top, MT and so on, which does not change

depending on the specific element type that the stack is using. And if you look further the you will find similar commonality with queue use the task scheduling process scheduling requiring queue user several problems that need list like implementing stack queue all those then any kind of object collections and so on and so forth.

The class templates are a solution to such code reuse where, the basic we identify the generic part of a data structure where you have the same interface and same or very close algorithms, similar algorithms but the implementations need to be different due to the element types can we combine them in terms of a common generic class template.

(Refer Slide Time: 04:50)

So, just to illustrate this is a left and right, if you just look in here this is a stack of character, which is character. These are just shown as comments the particular code lines which need the knowledge of the type and this is a stack of integer, so these are the lines which need. So you have integer here, char here, you have char here, you have char here, and int here.

(Refer Slide Time: 05:25)

Other than that the rest of the code is exactly the same so why not we replace this by a type variable like we did in case of the function. This is what leads to the class template which is parameterized with type and may have parameterized member functions. Rest of the definition is for the details and will look at the example.

(Refer Slide Time: 05:46)

So for the stack all that we do we parameterized this type element type ST. As you do that as you we can see the places where you need is when you push I need to know the element type which is T when I do a top I need to know the element type, pop does not need to know it, empty does not need to know that. Since this type is T I parameterized and exactly the way I had done in case of function I put a template here in terms of template class t saying that this is a template variable here. And that template variable is used in terms of these member functions.

So, this is what makes it a stack which is templatized, which can be instantiated for anything. Of course, for this stack template to work we will need the type T, the type variable T to satisfy certain property certain traits. For example, item is of type T and data i is of type T any data element. So we see that there is an assignment possible here. The copy assignment operator must be possible in this place without that you will not be able to instantiate the stack with a given particular type.

(Refer Slide Time: 07:14)

If we look at this now using this assuming that all that goes into the stack dot h the header, then I just instantiate it much like the way we are instantiating the function we just instantiate it say it for the character. This will now give me a stack of characters

which I can use this I will not go into explaining this code we have seen this number of times in this codes, we can use that stack to actually reverse the strain.

(Refer Slide Time: 07:48)

With that same header stack dot h I can now write a separate application. So, this same header that is a same templatized stack code I can write a different application to evaluate postfix expression. Since expressions here are of integers so I need a stack which will keep the expression values which of integer, so this is instantiated with int. If I had done with C, I would have required to use, two different stack implementations, the char base implementation for the reverse string and the int reverse implementation for this particular postfix evaluation problem, but I have managed with the same templatized stack definition and just instantiated with two different types. That is the basic power of the class template and this gives us lot of generalization in terms of the data structures in particular and different utility classes.

(Refer Slide Time: 08:41)

Now, naturally as I have mentioned that when we do this instantiation. Earlier we saw it for function templates now you are seen it for class template, we will have to make sure that the parameters the type parameters that are used in the template they will satisfy certain properties that is they may be of any type. Maybe other parameterized types also, they may template type themselves, but what is important is they must support the methods that are required for the implementation of the function template of the implementation of the class. So, (Refer Time: 09:21) traits like, they may require to support constructor, they will require to support different operators and we saw instances of that. Those are the basic type traits that both the function template as well as the class template will need to follow.

(Refer Slide Time: 09:34)

So, this is what you saw in case of function template, this is just for your recap.

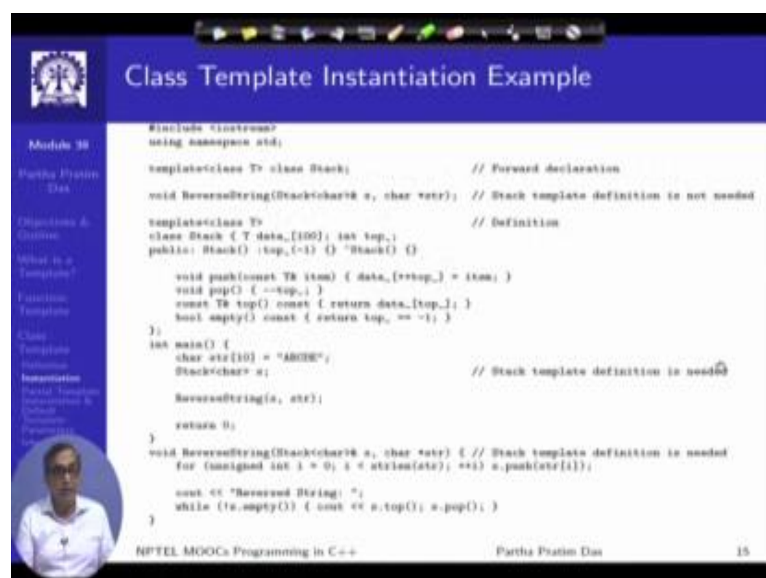
(Refer Slide Time: 09:42)

In terms of class template usually the instantiation needs to be done explicitly and it is important that since as a class it is quite possible that, I can define the class as a forward declaration without actually providing, so I can actually just write this which is called an

Incomplete stack type. It just says this is the forward declaration to tell the system that there is a class called stack which is parameterized by type T, but it does not say what the methods are and so on. So, with that if I try to instantiate the object then I will get an error, because certainly if object cannot be instantiated unless I know the constructor, destructor, other operators and members and so on.

But I can still define a pointer to this type; I can define a reference to this type. So I can define a reverse string function which takes this type as a reference when I actually do not know what that. But once I want to implement the body of the reverse string function when I want to use the stack operations naturally I will need to know what that stack definition actually is.

(Refer Slide Time: 11:09)



In terms of instantiation this you know lazy instantiation is something which is often very useful. So, I am just showing the same reverse string code in a little bit different manner, earlier this whole stack class was put into stack dot h enters included here, so as if the whole think was happening there itself. But now I am including it here to show if few things, for example, here we have a forward declaration, so with that forward declaration I can have a signature of the reverse string function which will reverse the

string put in here and because as a reference all that it needs to know that it is using a stack which is templatized by T and the template is instance is char in this case.

But it does not know what that type is, what it does not know how the type is implemented and it does not care because it is just looking at a reference. If I have this in my main I can actually invoke this function because all that I need to know is a signature of the function the body can come later on, so I have deliberately put the body at a later point of time just you show that the main does not need to know the body.

But certainly, this needs that I pass the instance of a stack as a reference parameter here. So, main needs to instantiate this stack. While you could define the signature of reverse string without actually knowing the definition of the stack you cannot write the main function, because you cannot write this instantiation unless you know the definition of the stack. So the definition of the stack has to precede the instantiation of the stack. Because now if you have an object instance we must be able to construct, it must be able to destruct, it must able to invoke all different operations.

So this is the kind of, I just wanted to highlight that in keys of a class template instantiation it is not necessary that you will have to always instantiate everything together. If you are instantiating the reference to the class or a pointed to the templatized class then you may not need to know the whole definition of the class, you could just manage with the declaration of the class a forward declaration which says that these is the templatized class what are the different types and so on.

(Refer Slide Time: 13:47)

This is the basic class template. Next we show something, this is just for your understanding of completeness, I will not get much in depth. This is just to show that it is like in terms of function template we saw that if the max function had one type parameter T and for char star we wanted a different behaviour so he specialized that and replace T and just put char star and put a different one function definition for that. This is also possible for class and I show that it is actually possible that if I have more than one parameter then I can partially specialize those parameters, so that is what I am trying to illustrate here.

So, there is the template where the student class here which is templatized by two types T1 and T2. T1 is a type of roll, T2 is the type name so possibility one could be the role could be an integer it could be a string and so on. Name could be a string type in C++ or it could be a char star C string type and so on. These are two different types that we have. So, what you do in that is basically, I am there is not much functionality given, you just do a construction and there is a print in which you can print these two fields, so just for illustration.

Now, what is interesting in the next one, where we actually partially specialize this? There are two parameters T1 and T2 and I have partially specialize this, I still continue to

have a template which has a parameter T1, but T2 have explicitly put as char star and then I have used. In case of T2 I am using char star I have explicitly put that char star. So this becomes the partial instantiation of the template. Earlier this template of student class needed two types to be specified T1 and T2, this needs only one type to be specified which is T1 the other has already been specialized. In a template definition, when you have specialized for all the type parameters then you say that the template is fully specialized otherwise you say this is partially specialized.

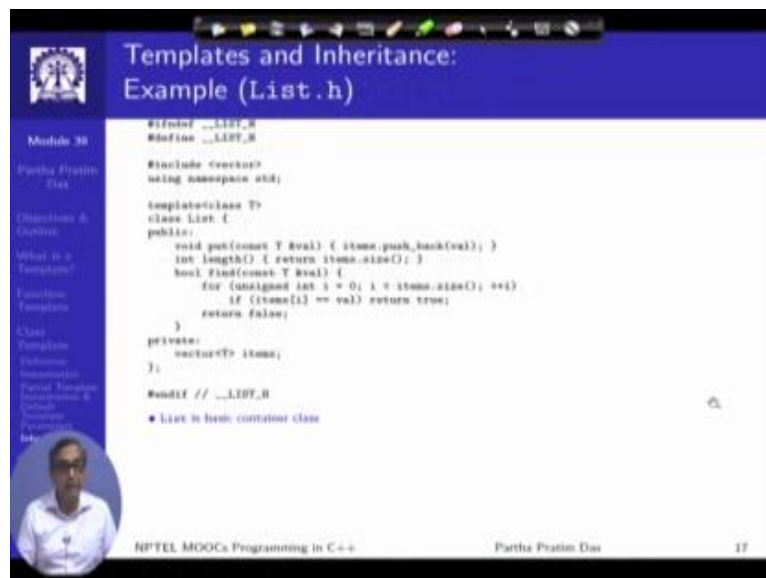
If we look at this some class instances with that, suppose we have created an instance like it in string naturally this specifies both of them. So it is trying to talk about this template with T1 being int and T2 be string. You can easily make out from the output, you have done a S1 dot S1 is a object created for this template version and we are doing S1 dot print that choose version one is being used.

In the second one, what we have used? In the second one, we had using something interesting. In second one what you are saying is, look at this carefully, in this we have also specified something as T1 equal to int or T2 equal to string. Recall that in terms of functions we can have default parameters of functions which are values we can write int x equal to initialized with 5, so that if I do not pass that parameter it will be taken as 5. Similarly, you can have default type parameters. So if I do not specify this then it will be taken as that type. If I am just saying string student int, if I say student int then it means that I am taking this. And I have not specified T2 which is taken by default to be string. So this is by default taken to be string.

I can do this, which will again mean this, where both of taken default parameters values. Default T1 is int, default T2 is int. I can do a student string then I do student string it means that I have done T1 to be string and T2 is a default which is also string. You can just see I have shown the output that you get generated. At the end, if I say that what did we do here? We said that the second parameter is partially specialized to char star. So if I put int char star then it does not mean this template because the second parameter has been specialized partially, so if I say this then it means this template and you can see we are printing for S5 when we do print S5 dot print you can see that the version two is being printed which shows that you are using the partially specialized template version.

This is just to show you that these kinds of things can be done you can have multiple parameters and partially specialize them as you go from one to the other and you can have default parameters also.

(Refer Slide Time: 19:39)



Finally, before I end I just show you an example of using template with class inheritance so this is just trying to create a bounded set kind of data structure. There is a list, there is nothing specifically interesting about the list is just a list which has a add function called put, has a length function, has a find function to find a particular element, uses vector internally to keep the elements. So that is kind of a supporting data structure.

(Refer Slide Time: 20:13)

Then you define a set using this list. A set has a list of items a class T. It has a virtual function that can add elements find the length and find. So what will basically do is if you add an element it will go to the list it will go to items and do a put. If you want to this you will do a do a put here, I think did I miss anything no. It will add does this, add actually if you want to add to a set, now this is a set, this is interesting, this is a set so every element has to be unique, set has uniqueness.

The way I do it is, I first take the value on this list I find out if the element belongs to this list if it does belong then this already there in the set so you do not do anything you just return. If I does not belong then the control comes here, then you put it that is add it to the list so this is what. Length is simply a rapper on the list length, find a rapper on the. This gives you a type of having set for any element type.

(Refer Slide Time: 21:40)

Now, suppose I want to have a bounded set. Bounded set by the name here, which is a set having two limits the elements will have to be within that limit there a bound set will be able to have only members which are within min and max values. So it is a specialization from the set. This you can see how you write the spec. The bound set is also templatized because it has a template parameter which is an element type, but it specializes from set T. Then the bound set, it has a constructor, it has an add. And, what simply does is when you try to add, it has to change that if is already there, it will return which is the behaviour of the set.

But, if the value is within the min and max then it will go to the parent set class object that is base class object and add that. But if it is not then it does nothing, it will simply ignore you can through an exception here also and do some other behaviour. But what I am trying to just show is here I have a bound set which is a specialization of set this is templatized, this also is templatized and this has a part of it as a component is has a list which as a part of it has a vector. All of these are templatized, I finally get a bound set which of any type using all these templatized classes and using the inheritance on this templatized class. So, this is how the templates can be mixed up with inheritance feature as well.

(Refer Slide Time: 23:38)

Templates and Inheritance:
Example (Bounded Set Application)

This is a final bound set application you can just complete the application and check and taken run it that you have added some numbers to this set, and then you try to find four which you are expected to get. And you check if you have of something like 0 or 25 in the list which should not be there because your list is between 3 and 21. So you say there is no unexpected value. This is just shows with the example as to how your bound set data type will work.

(Refer Slide Time: 24:17)

Module Summary

- Introduced the templates in C++
- Discussed class templates as generic solution for data structure reuse
- Explained partial template instantiation and default template parameters
- Demonstrated templates on inheritance hierarchy
- Illustrated with examples

NPTEL MOOCs Programming in C++ Partha Pratim Das 21

To summarize, we have introduced templates in C++ and we have discussed class template has a generic solution to data structure. Combined with the function template this gives us a major advantage in terms of being able to write generic programming, meta programming code and gives a foundation of what is known as a standard template library or STL of C++.