FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture41

Lecture 41: Introduction to Standard Template Library

Welcome to lecture number 41. The new chapter is called the Standard Template Library. Often, it is called STL. So, again, I am coming back to C++, right? So, now you are more familiar with C++ and Java.

And in the last one or two lectures, I talked about Python. So, my advice is you start using Python as well, right? So, when I am explaining the Standard Template Library, again, I am switching back to, let us say, C++. So, in fact, STL is nothing but a collection of C++ template classes and functions, right? So, we extensively talked about template classes in the last chapter, right?

So, here, STL is nothing but a collection of C++ template classes and functions. So, in fact, it provides data structures. Suppose you want to store data, right? So, there are several ways. So, one particular data structure that you have studied is a list or arrays, two-dimensional arrays, right?

Advanced version when we go, so you have a linked list, doubly linked list, binary tree. So, these are all the data structures. So, STL provides these data structures. So, you call it as containers the inbuilt all right. So, not only the data structures and you have the algorithm some of the algorithm all right.

So, that means so because of data structures and algorithms. So, this STL which is facilitating efficient and reusable code. So, STL which is built on three components. major components right which are nothing but containers iterators and algorithms also so when you are having a stl so basically it is defined to offer a standardized and extensible framework right for managing a data and performing the operations on it right

so these are all some of the characteristic features of this stl standard template library So, what are all the advantages? So, we can have code reusability since you know templates all right and then we have talked about the templates functions and template classes. So, the STL we have the code reusability that means, we can use it for any data types and performance is highly optimized because it can use many ah inbuilt functions. So, therefore, it is highly optimized and you no need to manually implement certain algorithms, certain operations.

So, performance point of view, STL is having advantage and consistency. So, since it is offering a unified framework. This ensuring the consistence in the interfaces across containers and algorithms extensible. So, you can extend the algorithm right. For example, assume that you have insertion algorithm in one of the data structures.

So, based on this insertion algorithm I can use this right as module can I extend? Yes, you can do this right. So, that option is available this is added advantage right. So, the extensibility. and reduce development time since you can use many data structures and algorithms readily available right.

So, which is already available. So, that means, the I mean you no need to develop anything new for example, the standard algorithms or standard data structures right. So, this is reducing the development time. So, these are all some of the advantages and as I said the three major components the containers iterators and algorithms right so the containers suppose i use vector list right set and map all right

So, that means the containers, which are nothing but storing and managing the collection of data, right? So, you can do it very efficiently. That is called containers. So, vector is a container, list is a container, set is a container, and map is a container, right? So, these are all examples of containers. Similarly, you have iterators: begin, end, rbegin, rend. So, that means it is providing an abstract interface, right. So, you can traverse all the elements in the container. For example, you are given a vector. You can traverse vector 0 to vector n minus 1, or list, or set, right.

So, you can traverse, which is the begin, which is the end, all right? Because sooner or later, we are going to talk about some data structures. Let us say doubly linked queue, right? Which is front, which is the back, all right? Or end, which is the end, right? Begin and end, correct. So, that is nothing but the iterators. Right, and algorithms. So, several inbuilt, for example, I have an algorithm for searching. Given a data structure, I want to search an element. Or, given a list, I want to do the sorting, right? Sorting in ascending order or sorting in descending order.

Or, I have to update the data or modify the data. So, these operations are already, right? Several inbuilt functions are there in STL, all right. Suppose you are working as a developer, all right. So, assume that you do not want to know the deeper details of the algorithm, but you want to use. I want to, let us say, sorting. I want to use a sorting algorithm, right.

So, there are several actual data sector when you study you have several right algorithms and you have to implement, but here what you can do you can call sort right the sort function that you are calling and then it will do the sorting. You can see how you are going to sort whether it is a increasing order or decreasing order it is available or finding right finding the elements in the list. or finding the elements in the vector all right or transform you are transforming into some other data modifying the data. So, these are all possible so that means, there are three that we are seeing in this particular slide it they are the major components of STL containers, iterators and algorithms. So, in fact we are talked about containers in python also if you recall

right there we talked about the instances right. So, here in the C plus plus is nothing, but the objects containers are nothing, but the objects right. So, the objects which are storing the collection of data if I take an array object right int a, a of 10 that means, it is storing a 0 to a 9 all right. So, it is storing the collection of data. So, in a similar way the containers they are nothing, but the objects that is storing the collection of data.

And here when I am talking about STL containers. So, this provide the various range of data structures right. So, starting from the dynamic arrays to hash tables right. So, STL containers you have various levels of data structures right. So, it is it is giving it is providing right.

So, dynamic array to hash table. So, we can see in STL containers. If I am talking about containers, every container type is useful or designed for specific use cases, right. So, this balances the trade-off between memory usage and algorithms like insertion, deletion, and traversal speed, right. So, you have memory usage, and then assume I am using one or all of the algorithms.

Insertion, deletion, right, and traversal speed. So, it balances the trade-off between memory usage and dictionary operations like insertion, deletion, or traversing, right. So, containers are nothing but the foundations of STL. So, which enables efficient data manipulation and management, right. So, efficient data management and manipulation.

So, what are containers in STL? Right. So, as we have already seen, it is nothing but a data structure that stores and organizes a collection of objects. Right. So, which is the standard definition.

So now, when I am talking about the generic. Right. We have used in the last chapter; we have used the generic concept. Right. So, STL provides generic containers.

Now, you know the meaning of generic. Right. So, this can work with any data type. All right. So, generic containers.

In Java, generics in Python, we have seen. So, that means STL is providing generic containers. So, that means it can work with any data type. All right. Also, containers manage memory allocation and deallocation.

So, these things will happen internally. So, that means, the program will be simplified. So, when you are having allocation and deallocation. So, you are safe with a memory management and these are all the broad categories right the containers. So, you have sequence containers, associative containers and unordered containers, right.

So, these are being categorized based on the structure and functionality. So, what do you mean by sequence containers? So, in the previous slide we had seen. So, explanation for this. So, when I am talking about vector, right or list or deck doubly ended queue, right.

So, these are all the sequence containers. That means, You can take list or an array, right. So, they are in the contiguous memory location that means a linear ordering, right. So, they are all in the linear ordering vector list or doubly ended queue or simply queue, link list.

So, these are all linear order. So, you call it as sequence containers, associative containers. So, if I take set and maps. So, it may store the elements in some sorted fashion and it is also having the keys. So, the keys you can retrieve fast.

So, the examples are set and map. So, it stores the elements. So, even if it is storing in any form you are having keys. So, with the help of keys you can retrieve it. So, you will find out in a sorted fashion setter map and they are all using keys for retrieving the elements quickly.

Unordered containers. So, in the case of unordered containers means any fashion the elements are in any fashion. That means it was not sorted, you call it as unsorted, right. And then it uses, these unordered containers uses hash tables, right. So, this unordered container, so they are using hash tables, right.

So, since the elements are in unsorted fashion, so when you want to retrieve, right, you use hash table. So, that means you can, if you want to retrieve the elements, so you can do it faster. So, what are all the examples you have unordered underscore set and unordered underscore map. So, if you simply use set and map they are ordered whereas, here in this case they are unordered. So, like this we can define we can categorize the containers sequence containers associative containers and then you have unordered containers.

So, as we had seen in the case of sequence containers. right and the elements are arranged in a linear fashion so that means obviously the some of the basic dictionary operations like insertion deletion and traversal can be performed very easily right so in fact you have the operations all right so these containers have the inbuilt operations so for example i mean i have a common sequence container such as a vector right vector is a dynamic array list is let us say double linked list and the doubly ended queue you call it as a deck right so these are all some common sequence container so already inbuilt functions are there

so that suppose i want to have some operations like a dictionary operations like insertion, deletion, updation right all these operations i can perform very easily right so in fact when you want to manage the collections of data right so the sequence containers are very ideal So, because you have a sequential access. So, for example, I want to search an element, right. So, sequential search.

So, starting from 0 to n minus 1. Yes, time complexity we will worry later, but I mean you can move around from 0 to n minus 1, right. So, it is ideal because the elements are arranged in a linear fashion. The sequence containers are ideal for managing collections of data, right. So, because of the nature.

right, the sequential nature. The next one is a vector which is a dynamic array, right. So, dynamic array we know that I mean you can resize the number of elements in the array. Assume that first time we are using for 100 elements to sort and then same array name we are going to sort for 50 elements or we are going to use it for searching, right. So, you call this as a vector which is a dynamic array.

So, that means it supports the random access and this is allowing efficient insertion and deletion at the end all right. So, you can do the insertion the basic dictionary operations and in fact the key operations that you can do over here push underscore back that means you are adding an element at the end all right adding an element to the end pop back. So, remove the last element right suppose I have 10 20 30. right I use pushback let us say 40 is a function I am inserting an element 40 right so that will add over here 40 right and assume that one more one more time I am calling pushback by passing 50

I will insert now pop back, so remove this. This will be removed. Pop back size. The size will return the number of elements, right? So, right now, the number of elements is 4. 50 have been removed. Right. So, this will return the number of elements in this particular vector. And at.

At is nothing but the axis and element by index. So, the 0th index is 10. The first index is 20. Right. At of 4.

At of 4 is at of 3. 40. Right. So, that means when you want to access an element by index. So, you can do this, all right. So, suppose you are writing a program where you have to insert a lot of elements, right.

So, in that case, it is advisable to use the vector, which is nothing but a dynamic array. So now, we will see an example. In fact, we can run the code also because, in the last few classes, we have been talking about Python and Java. Right. So, we will again maybe run this particular code.

So, let us see what is happening, all right, in this case. So, here, for the first time, we are using the keyword called vector, right? So, vector in STL. So, what do you have to do? You have to include vector: #include .

So, let us see all the uses. You see, int main. So, here, vector is a class; in fact, it is having the integer. Correct? So, the array, which will be having integers, is a dynamic array. The dynamic array of integers.

That is the meaning over here. And then you are using the name of the vector is numbers. Right? Name of the vector is numbers. So what you are doing?

Numbers push underscore back. So first I will put 10. Right? So push underscore back. First I will put 10.

Right? Again I am calling. Members is calling. Sorry. Numbers is calling.

Push underscore back. Right. You are pushing another element 20. Numbers dot push underscore back you are entering another element which is 30. Right.

So now 10 20 30. Going to line number 13 for int i is equal to 0 i less than numbers dot size. We are calling the function size function. All right. So 1 2 3.

So in fact it is 3. So that means for i is equal to 0 i less than 3 i plus plus or plus plus i. All right. So cout. Element at index i. Alright. So element at index i. So i is 0.

It is printing i. That means 0th index. Numbers at i. You are using another function. So what are all the functions you are using? Under vector. Push underscore back.

Line number 8. Line number 9 and 10. And line number 13 you have size. Numbers dot size. And line number 14 you are using at.

So here numbers at i 0. Numbers at 0 is what? 10. So the first output we have to get 10. Again when i becomes 1.

Numbers at 1 is nothing but 20. And numbers at 2. What is that? 30. So when you are pushing these 3 elements.

This is output. I will just put OP. OP is nothing but output. Alright. So up to here.

Over. STD endl. So up to here it is over. Now line number 18. Numbers dot pop back. Back.

Alright. The end you have to pop. So, the end of the element is 30. This has to be popped. Going out.

That is the meaning of pop. Alright. Deletion. So, what will be remaining? 10 and 20.

The remaining will be 10 and 20. So now. When the remaining is 10 and 20, what is the size? I am printing numbers.size, right. How many elements?

Two elements are there. Two should be printed. So, these are all the output we will get. So, in fact, when we run the code, so we are getting 10, 20, 30 and 2, right. So, let us see in the C++ prompt how

This is working. Let us go to the compiler. This is what I explained, right? So, when we run the code, compile and run, yes, you can see the output over here, right? So, 10, 20, 30 we got and then when you put the pop back, 30 will be out and the number of elements, once 30 is out, we are getting the output 2, okay?

I hope it is clear now. List, right? The next vector we had seen, list. So, list is nothing but the doubly linked list. right.

So, a doubly linked list is what is happening. The structure will be a node like this, and it may have some value. And then there are two pointers: one pointer is pointing in the forward direction, and another one is in the backward direction, right? So, this will have, let us say, 20, right? And then this may point out over here, right? And then maybe forward. So, that means one link is forward, and another one is backward, right? So, you will have data

and then the pointers. So, one pointer is pointing inward, and another pointer is going outward, all right. So, this is happening in two directions. So, usually, a singly linked list will be like this, and then the link will be like this, with only one direction pointer pointing in one direction. So, whereas here, you have the next as well as the previous.

So, you have the information and the pointer. Just point to the next node as well as the previous node. So here, if you have a next node, the next node will point here. Whereas the previous node is pointing to 10. Let us say another node I have is 30.

So 20 pointing over here. That is the meaning. So you have one pointer is pointing to the next. Another one is the previous. So that means you will have the efficient insertion and deletion.

Because you have two pointers better than singly linked list. So you have the advantage. And moreover here you have several inbuilt functions. So, that means it is supporting efficient insertion and deletion at both the ends right or in the middle. So, the main operations here you have push front, push back, pop front and pop back right.

So, these are all almost similar one we had seen vector right, but here you have the advantage of two pointers one pointer is pointing towards the forward another one is backwards Or the previous next. Right. So that means push underscore front. It is adding the element to the beginning.

So, suppose I have. The doubly linked list is like this. 10, 20. Right. And then it is going forward.

And 1 will be approaching here. Right. So, what will happen? I mean, it adds an element to the beginning. Right.

So, that means this pointer will take over. And 5 will be added. Right. And then it will have a pointer towards 10.

Initially it starts from 10. So when I put push front, 5 will be added like this. Or you can add push back. 25 I want to add. Alright.

I want to add let us say 25. So I can add it here. That is what? Push back. So one pointer is going here.

Another pointer will go. Alright. Pop front. So pop front is nothing but I will delete the front one. Pop front, right.

So, deletion of this node, that is pop front and pop back, I will delete this node, right. So, these functions, in fact, these are the inbuilt functions. So, in fact, if you study the separate course called data structures, you will have a doubly linked list there. So, you have to write all the code, right. Whereas, in the case of STL, all these you are having as an added advantage.

So, that means the inbuilt function is there, right. At one point of time, you know, it is advisable you have to write the code because when you When you are developing an algorithm, you may have to do the efficient algorithm, right. So, all these concepts you will study in data structures, right. So, right now we are

enjoying with these functions. So, because these are all available and I can call. So, once I define, let us say, doubly linked list, I can have what is meant by push front, what is meant by push back, I simply call. So, I have told you what exactly it is happening. So, these are all the key operations that you can find in the list, and you call it as a doubly linked list.

Right. So, let us consider this program, all right. So, you have to include iostream first. I am including list, OK. So, the list of integers. So, let us give it a name as numbers, all right.

So, list followed by the angle brackets, put int inside. Now, the numbers push back initially. It is empty; the doubly linked list is empty, all right. So, push back 10. So, first it is 10. Right.

It will be like this. Correct. So, next push back 20. Right. So, 20 push back.

So, 20 will be like this. Maybe null. Here it is null 20. Right. And then push front.

So, 5 will be here. This is what the meaning of line number 8, 9 and 10. Yeah. Of course, the pointer will go. So, right now here it is understood that this point is pointing to null right this pointer and similarly this point is pointing to null ok.

So, here it may be P and here it may be Q because one pointer should point to the previous right another one is the other way around. For example, here I put Bose in a same direction P should be like this right it is called the right pointer and Q is the left pointer and here once it is like after push front this will be obviously pointing to null and this will be pointing to null. In fact, the course called data structures will study all this. So, right now we are utilizing all inbuilt functions.

So, when we are calling this, so this is happening. Right now the data structure is like this. So, now for int num right numbers, numbers is a list mapping into num that means the both the base addresses are pointing to the same location. So, now cout num, right. So, that means it is printing all the numbers.

For example, here in this case, it will print 5, 10, 20, right. So, it will print 5, 10, 20, all right. And then see out, it is just a dummy statement. See out, nothing will happen, maybe a new line kind of. And then pop front, right.

So, when I do pop front, I am removing 5. Let us say 5, 10, 20. So, here I can use it. So, pop front meaning is I am removing 5. pop front right now the remaining array is 10 and 20 and then pop back line number 19 you have popped back

so that means this will also be popped out what is remaining 10 right so the remaining 10 correct so that i am going to print num correct so when i run the code i will get the output 5 10 and 20 correct i got 5 10 and 20 and here pop out remaining will be 10 right. So, that will also be getting printed right. So, this is all about your list.

So, when I run the code I will get 5, 10, 20 and 10. In fact, it is a doubly linked list right. So, this is called doubly linked list. So, this is all about list and in the next lecture we will talk about the doubly ended queue you call it as deck ok.

So, from here I will start in the next lecture. Thank you all.