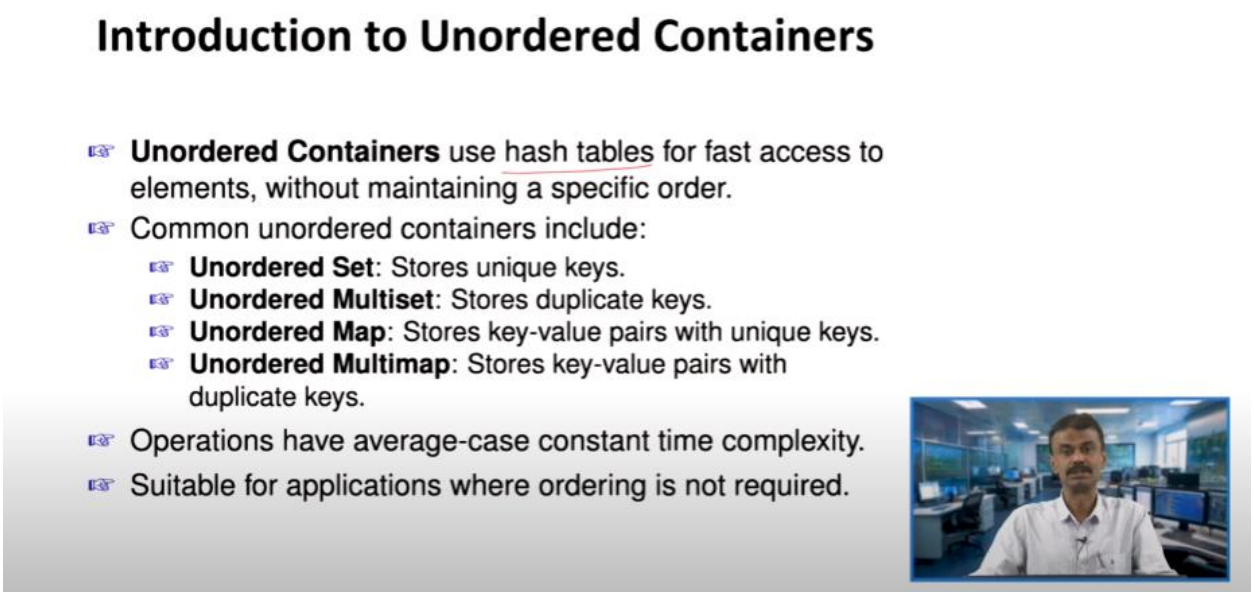
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture43

Lecture 43: Unordered Containers, Iterators



So, welcome to lecture 43, the Standard Template Library, called STL. In the last lecture, if you recall, we talked about map and multimap, all right? Also, we saw set and multiset. So, we completed map and multimap, and in today's lecture, we will talk about unordered containers. I will start with unordered containers. So, what do you mean by unordered containers?

So, the unordered containers include unordered set, unordered multiset, unordered map, and unordered multimap. So, these all use hash tables. So, in fact, these unordered containers use hash tables. So, when you use hash tables, you can have fast access, alright? So, I mean, you might have used the mod function, right? The mod operator.

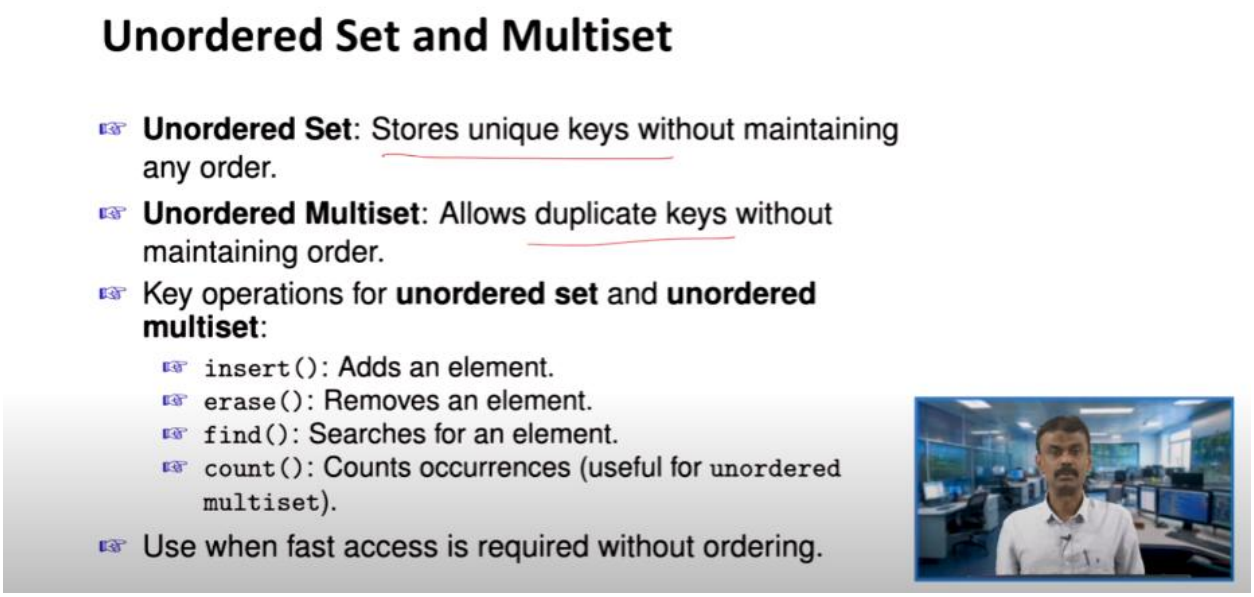
So, based on the mod operator, you can define the function h(x) with the help of a mod operator, alright? n mod k, or something like that, or x mod k, okay. So, something like that. When you use the function, it's called the hash function, right? So, with the help of a hash, So, I'm not going into detail. Just check how we can use the hash, for example,

the mod operator, when you have only a set of 11 elements, alright? So, it will keep on moving. Correct? A set of 11 elements assumes that the array or the list contains only 11 elements, correct?

So, from the 11 elements let us say I want to insert 65 right. So, what is 65 mod 11 all right. So, it is nothing but 10. 65 mod 11, 11 into 5 is 55, reminder is 10. So, the 10, I can insert this key 65 in the 10th location.

So, the hash function I am defining key mod the number of elements in the array, correct. So, H of key, I call it as a H of key, that is a function. So, that is called the hash function. So, this unordered containers they are using hash table. So, this is for the fast access.

So, for example, I want to search an element 65 in the array of 11 elements. So, what I will try to do I will find out 65 mod 11 and then the 10th location I will reach directly right or I can access 10th location directly. I will find whether 65 exists or not. If it is not I will keep on moving right. So,



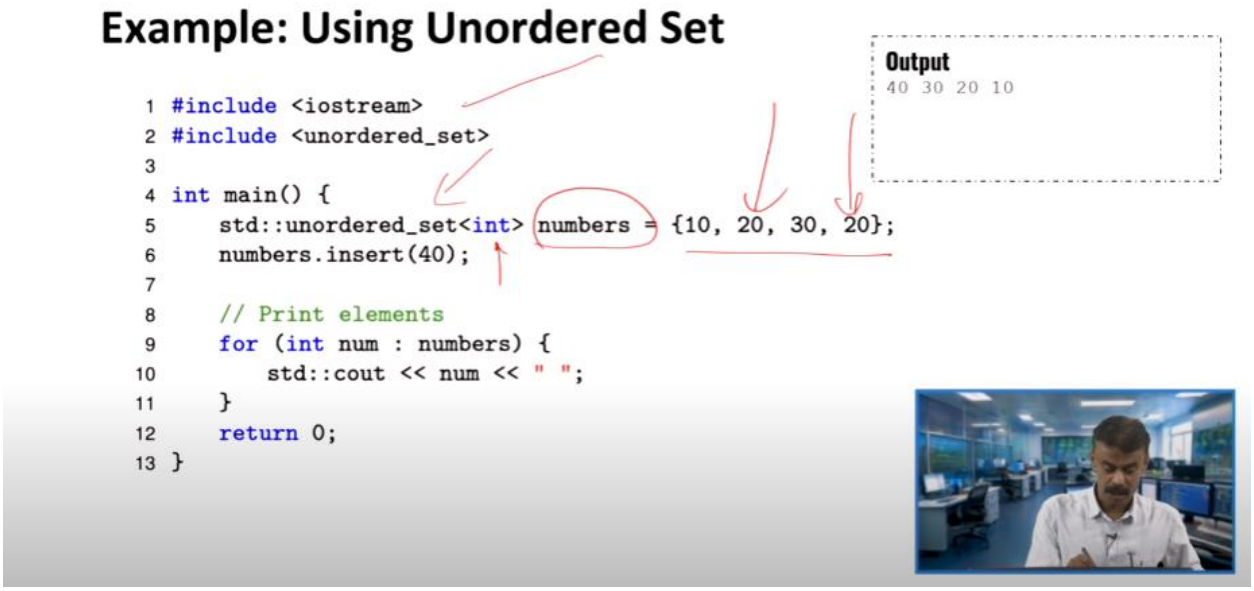
Compared to your sequential search of an array, this is slightly better, right? So, that is called the hash. So, in this case, the unordered containers use hash tables. So, this is mainly for fast access to the elements, and it will not use any specific order, right? So, without maintaining, obviously, I can directly go to the 10th location, correct, when in the case of key 65.

So, the common unordered containers are nothing but the unordered set. That stores unique values or unique keys. And unordered multiset stores duplicate keys. Unordered map stores key-value pairs with unique keys. And unordered multimap.

So, these all store key-value pairs with duplicate keys. So, the operations. So, whatever dictionary operations we can do. The average case has constant time complexity. So, if the number of elements is n, I can do it in order k, where k is the constant steps, right?

So, this is suitable for applications where you do not require any ordering, right. So, that is called unordered. So, now initially we will start with unordered set and multiset, right. So, the unordered set stores unique keys. without maintaining any order, right.

It stores unique keys without maintaining any order, whereas unordered multiset allows duplicate keys as well and again does not maintain any order—that is why it is called unordered, right. So, these are some of the functions or operations in the unordered set and unordered multiset. So, you can have insert, right? Insert function adds an element; erase removes an element. Fine, searching for an element and count, all right. So, this counts the occurrences.



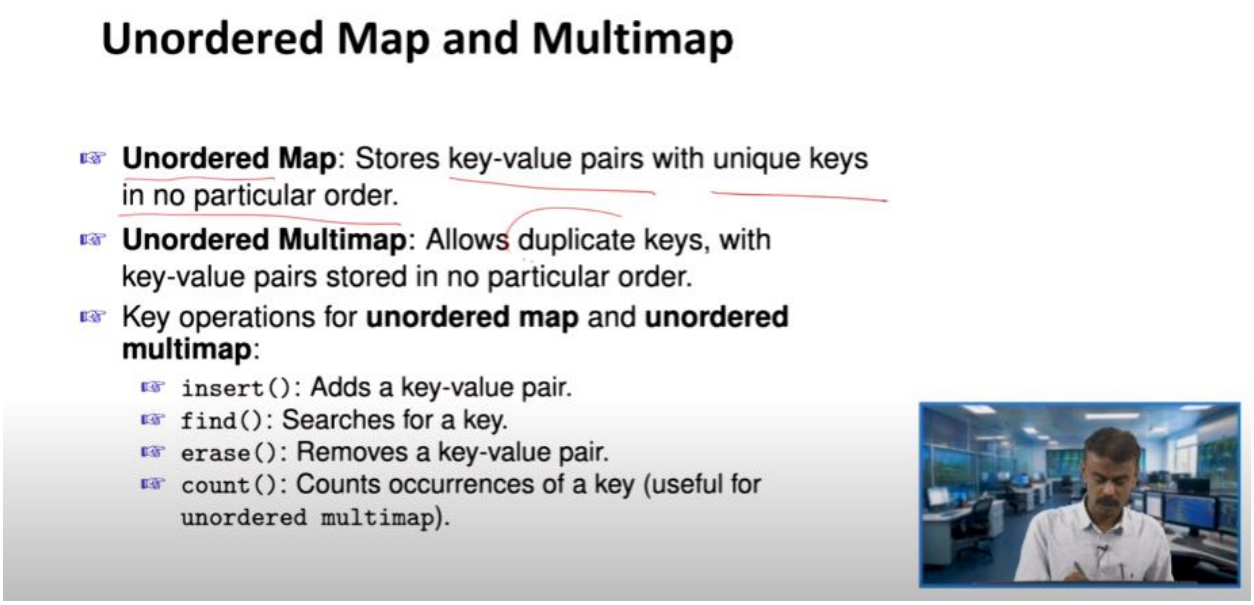
So, when we are using unordered multiset. So, this counting is useful whenever you have some data without any order. So, this unordered set and multiset They are useful for faster access. So, let us take this particular example, all right.

Now, an unordered set of numbers—that is an object, correct. So, here you can see, all right, the definition of an unordered set: it stores unique keys without maintaining any order—unique keys. But here, we are finding out the duplicate key, which is why I have gone back to the previous slide. So, here we have duplicate keys: 20 and 20, right. So, that is the initialization we are taking for the numbers. Unordered set is a class of integers, right? Unordered underscore set is a class that is under—you are including unordered set, fine.

So, now, this numbers—the object numbers—is invoking the insert function, a member function, and you are inserting 40. So, when you are inserting 40, 40 will be added. In fact, the set contains 10, 20, 30, 40, but I do not know in which order I am going to get the output. So, here you have numbers. So, numbers—it is mapping to num.

So, that means both addresses are pointing to the same location. And then, right—in fact, it is num 0, num 1, num 2, num 3—so that will be printed, right. So, when I am running the code, you can see I am getting the output: 40, 30, 20, and 10, okay. So, since the first time I introduced unordered underscore set, So, let us see how this is working in C++. So, we will go to the code, right? And next one: unordered set.

Yes, here you go. Numbers, unordered set. I am inserting 40. So, you have 10, 20, 30, 20, and then we are inserting 40, same example. So, when I am executing, compiling, and running the code. So, what is the output I am getting? I am getting 40, 30, 20, 10, right?



So, this is unordered. Usually, the ordered one we know, ordered set, we know it was giving 10, 20, 30, and 40, all right. So, what we can do is randomly insert one more element. So, let us say between 10, 20, 30, let us say 25, right? So, let us say 25, yes, and then if we add, let us see what is the output it is given. So, we have included 25.

So, you can see 25, 30, 20, and 10. Right. So, this is a good example of unordered, right? Because the previous one was following descending order. So, you may ask the question: will it give a descending order? But this example is giving that. So, the first element is 25, then 30, 20, 10.

So, this is a perfect example of an unordered set, but one thing you can find is there is no duplicate. 20 exists two times. But you can see the output: 20 has been printed only one time. So, the next concept we will see is unordered map and multimap, right? So, the unordered map. So, unordered, right.

Example: Using Unordered Map

```cpp
#include <iostream>
#include <unordered_map>

int main() {
    std::unordered_map<std::string, int> ageMap;
    ageMap["Alice"] = 25;
    ageMap["Bob"] = 30;

    // Print key-value pairs
    for (const auto& pair : ageMap) {
        std::cout << pair.first << ": " << pair.second << std::endl;
    }

    return 0;
}
```

Output

```
Bob: 30
Alice: 25
```

The keyword itself is unordered. So, this stores key-value pairs. And then you have unique keys, but it does not follow any order, right. Whereas, in a map, we had seen the Alice-Bob example; it follows a certain increasing order. Right, a certain order, whereas here it does not follow any particular order.

So, you call this an unordered map, right? Similarly, you have an unordered multimap, which also allows duplicates with key-value pairs stored in no particular order, all right. So, here, duplicate values are allowed, whereas in the previous case, duplicate values were not allowed, right. So, whereas here, in this multimap case, duplicate keys are also allowed. So, what are all the operations we can see?

As usual, we can see the insertion. So, you can add a key pair value, all right. So, key value pair and we can do the find, all right. We can use find operation or find function which is searching for a key and erase which is removing a key value pair and count, right. So, that is counting the occurrences of a key.

So, this will be useful for unordered multi-map. So, when you are using unordered multi-map, and then it is allowing the duplicates. So, we are finding out or we are counting the number of occurrences of a key. So, let us consider this example unordered map all right.

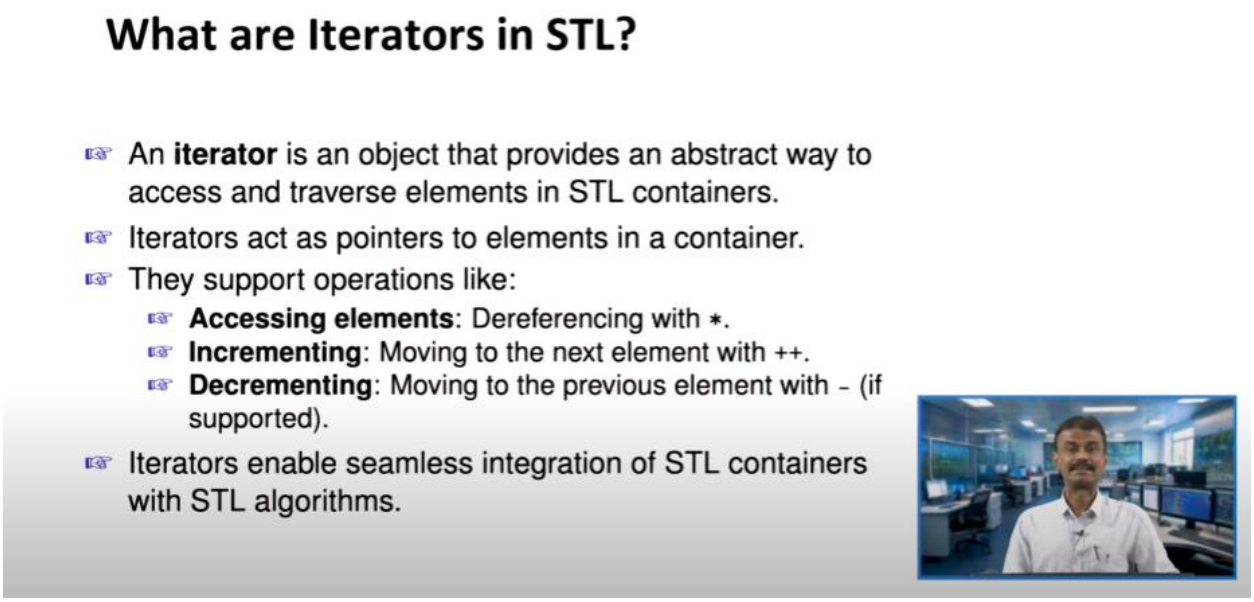
So, I include unordered map as a header file and under unordered map. So, which contains string and integer previous case we had seen integer when you have a set unordered set. So, string and integer and then you are having ageMap as a object.

Alright. So, we will take a same example what we have done in the case of a ordered map or simply map.

Right. So, ageMap of Alice 25, ageMap of Bob 30. Correct. And then auto address of pair that means the base address and base address of age map and pair pointing to the same location and when we are printing pair dot first and pair dot second. pair dot first is what right either Alice or Bob because now we do not know what order

it is following it is exactly like how we printed right 25 30 20 10 or 40 30 20 10 right so one of them will be printed right now assume that we do not know the output right so the first one will be the name and the age second one will be another name right because it is unique right another name and another age right so this will be printed right so in fact when we run the code we are getting the output bob 30 you can see here bob 30 we are getting first that's what i said which one will be printed i don't know if it is a ordered map

we know that we are going to get it in an increasing order that means alphabetical order Alice will be printed 25 that is what i confidently told the output if you recall what we have done in the case of map ordered map all right so here bob is getting printed first in this output right and then Alice bob colon 30 and Alice colon 25 in fact we can check it maybe we can add also some other name and then we can see what it is getting printed first we run the code so here you go we'll see the output yes bob 30 Alice 25 so what we can do we can include some other name start with let us say charlie



Age map, let us say Charlie starts with 25, 30, 35. We will put 35, all right. So, let us see how it is printing. Yes, you can see Charlie 35, Bob 30, and Alice 25. Okay, 30 and 25, yes, all right. So, it is not following any order. Not following any order. Since unordered multi-map, you can try with the same name, all right, and then you can check how the duplicate is being, all right.

Here, in this case, map if I put duplicate, it will not work, whereas in the case of Try with the duplicate, all right. Unordered multi-map also you can try, or we can go back to the same. What we will do, we will put Alice is equal to 35. Shall we go back to the program? Yeah. So, what I will do?

So, here I put, instead of Charlie, I put Alice. It is a map, all right. So, we know that the duplicate should not exist but we will see which output we are getting, right. So, it is the latest one; the old one is getting overridden, and you can see Bob equal to 30 and Alice is equal to 35, we are getting, right. So, you can try what I am giving as an exercise. You try with a multi-map, unordered multi-map.

In fact, we tried an ordered multi-map, right. In fact, I gave an example. Now, I am giving it as an assignment, right. So, in the case of an unordered multi-map, if you run the same code, can you tell me what will happen?

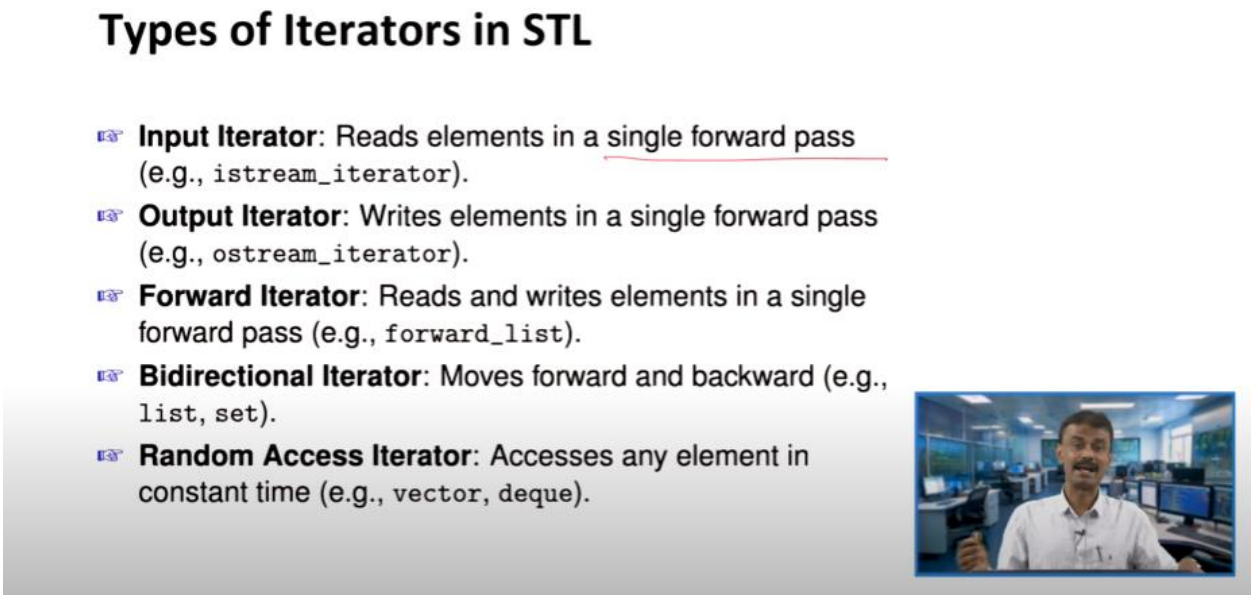
So, 2 times Alice will be printed in that particular C++ code that we had just now seen, ok. So, now the next concept: iterators in STL, right. So, what do you mean by iterators in STL? So, an iterator is nothing but an object, right. So, that gives an abstract way to access or, you can say, access and traverse elements in STL containers. Correct? So, we can access and traverse with the help of iterators. So, this is nothing but pointers. Right?

So, equivalent to pointers or acting like a pointers. Right? To elements in the container. So, the container let us say 1 to 10. Right?

Or 10 locations. So, this is acting like a pointer iterators are nothing but the pointers or you can say acting like a pointer to the elements in the container. So, what are all the operations they are supporting all right. So, you can access the elements that means you can dereference this with star all right the pointers and increment means you can move to the next location. So, when you want to move to the next location you use plus plus let us say iteration it all right.

So, if I put star it I can find out the value. I t plus plus I can move to the next location or minus minus it will move to the previous location right. So, these are all the operations the iterators can support that means accessing elements incrementing and the decrementing. So, accessing element or when I want to print the element I use star the value right the dereference operator. The increment operator plus plus if I use for one location it will move to the next location.

decrement operator one location it will decrement to the previous location right. So, when I talk about iterators. So, the iterators enable the seamless integration of STL containers with STL algorithms right this is another powerful applications. So, the I mean we can in fact, we can find the integration of STL containers with the STL algorithms. So, these are all the types of iterators in STL right the input iterator.



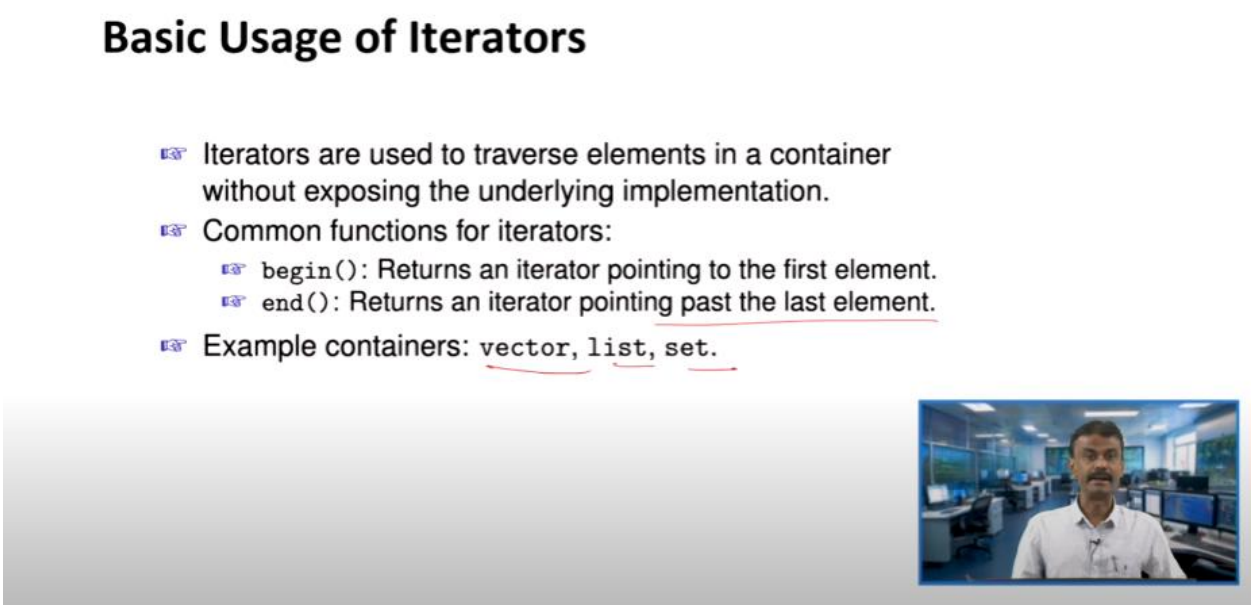
So, the input iterators that are reading the elements in a single forward pass right the single forward pass. You call it as a I stream underscore iterator this is an example and output iterator right So the reading. Now it's the writing. So the writing elements in a single forward pass.

Output stream. O stream underscore iterator is an example. So now if you want to both read and write. Elements in a single forward pass. So for that you have an example.

Forward underscore list. So you call it as a forward iterator. And then by directional iterators. Move forward and backward. So one pointer is moving forward.

Another pointer moving from the backward. So several examples you have in fact. you can use it for the list and set all right. So, for example, several algorithms like quick sort and all you have to move right forward and backward one pointer forward another pointer backward right. So, that is called as the bidirectional iterator and random access iterator.

So, you can access any element in a constant time. So, for that you have the STLs vector and deck right. So, in the previous case bidirectional iterator you have list and set. So, here in the case of a random access iterators you have vectors and decks.



So, let us consider the usage of iterators particularly the basic usage of iterators. So, particularly when you want to traverse the element. right in a container without exposing the underlying implementation all right so i want to traverse the complete elements let us say in a list correct and then i am not sure i mean what about the underlying implementation right so without exposing so you can do the traversing so the common functions for iterators are begin and end.

so it is returning begin which is returning iterator pointing to the first element correct begin and then end returns then iterator pointing past the last element So, it is pointing the last element end right. So, the containers the example containers are vector list and set correct. So, in these containers suppose I mean I am using the iterators. So, that is used to travel for example, I am using the container vector or let us say list.

Example: Using Iterators with a Vector

```
1  #include <iostream>
2  #include <vector>
3
4  int main() {
5      std::vector<int> numbers = {10, 20, 30};
6
7      // Using iterators to access elements
8      std::vector<int>::iterator it;
9      for (it = numbers.begin(); it != numbers.end(); ++it) {
10         std::cout << *it << " ";
11     }
12     std::cout << std::endl;
13
14     return 0;
15 }
```

Output

```
10 20 30
```

So, under list I want to assume that I want to work it out for certain sorting algorithm like a quick sort. So, one pointer I want to have it in the begin another pointer I want to have it in the end. So, that means it will be returning iterator pointing to the first element begin and when it is returning the iterator pointing to the last element it is called end ok. So, these are all the common functions where we can use it in iterators. So, now

We will take this particular example using iterators with a vector all right. So, let us include vector previously we have done and in the main program I have the objects numbers all right I have the objects numbers and the class is a vector of integers it contains 10 20 and 30. So, now how do you use the vectors right how do you use iterators. So, now under vector right iterator correct it is like std scope resolution operator cout right.

So, this iterator goes under vector of integers and that is a class iterator is a class and it is the object it is the object. So, now it we use it for loop it numbers dot begin right. So, the it pointer is pointing to the begin it is it pointer which is pointing to the address location of The begin, the first one, vector, the first element of the vector. So, address of this, not the value, value is 10, right.

So, the it is now pointing to, according to this, it is pointing to the address of the location where the element 10 stored, right. And this is the end. So, right now it is pointing here.

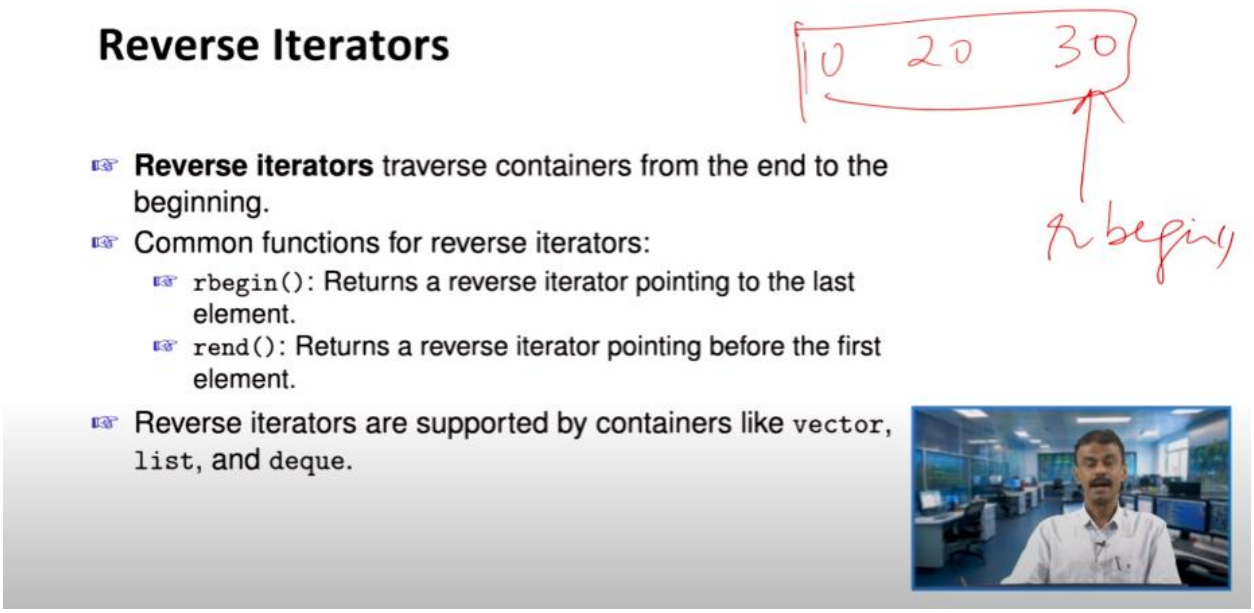
So, the check condition is whether it is equal to numbers dot n, right. So, if it is equal, so that is a stopping condition, stopping criteria.

So right now it is not because it is pointing to begin. It is not end. So begin cout star it. So it is pointing to the location of 10. So when I put star it, the value of that particular location.

So the value of that particular location is 10. So I should get the first output 10. So now plus plus it, this is what we said. So plus plus it, it will move to the next location. right.

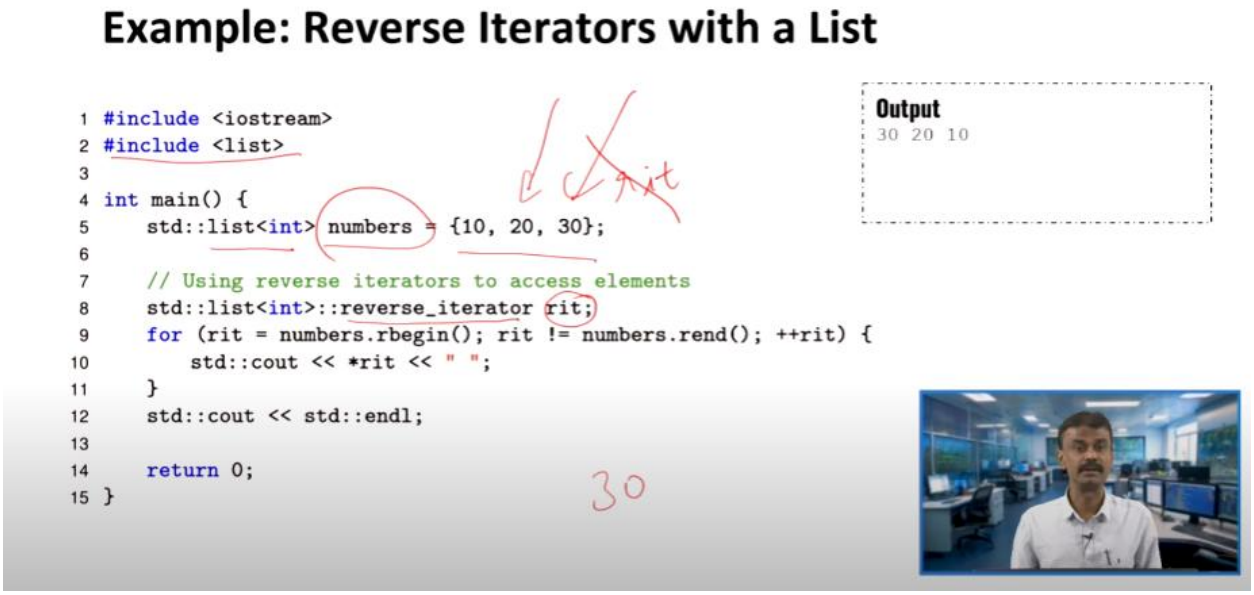
So, what is the meaning of next location? Next location is this where the value 20 is being stored. So, star it we will get 20 right in this location right star it is 20 right. Now again plus plus it. So, it is moving over here.

So, it is equal to numbers n, right? It is, in fact, plus plus it. We are going to print star it. So, 30 will be printed, right? And then your condition will be true, right? So, therefore, it will be coming out of the loop, right? So, it will be coming out of this for loop, right? Still, it is being reached. The values will be printed, all right? So, that will be printed under star it, correct? Right.



So, that will be printed, and you can see the output will be 10, 20, and 30, all right. So, when you run the code, let us go to the C++ code—I mean, the C++ compiler—and run the code. Iterator, yes, same example. So, you can see we are getting 10, 20, and 30, all right, yeah. So, this is all about your iterators, and similarly, we can have the reverse iterator, all right.

So, that travels from the end to the beginning. Right, the reverse way of traversal. So, in the reverse way of traversal, here you have the functions rbegin and rend—r stands for reverse. So, this is returning the reverse iterator pointing to the last element, rbegin, all right. So, suppose that the same example: 10, 20, 30, so your rbegin is this.



Right, the function rbegin all right and this is your rend that is what the reverse begin and reverse end and rend is so rbegin will return the reverse iterator pointing to the last element whereas rend that is returning a reverse iterator pointing before the first element right pointing before the first element ok. So, the reverse iterators are supported by containers like again same the vector list and deck so let us take the same example but in a reverse way correct so let us consider include list ok so here i am including list

So, numbers is an object, list of integers, the list of integers are 10, 20 and 30, right. So, now under list you have the class reverse underscore iterator and my object is rit, right, reverse iterator, r stands for reverse and IT is iterators, reverse iterator. So, now for

reverse iterator numbers begin. So, you have 10, 20, 30. So, it is beginning here, right, rit is here, all right, RIT.

This is not equal to, all right. So, it is printing star rit. So, what it will print? 30, all right. So, here plus plus rit, the reverse, all right.

So, therefore, this will move over here. That is a meaning reverse. If you put a IT, the case of iterators, all right. So, then it is a different story. If you put only numbers dot begin, all right.

Remove the word then it will be different story it is moving so here in this case it will move in the reverse direction correct so 20 right now it is not pointing so therefore it is moving all right so 20 is not pointing to numbers rend correct so therefore it is moving right so therefore it is moving plus plus rit and then it will print star rit that means 10 will be printed so now it is already there so rit is equal to numbers dot rend so therefore it will come out of the loop by printing 30, 20 and 10, correct.

So, initially when it is the numbers dot rbegin, correct. So, it is rit is not equal to numbers dot rend. So, therefore, first 30 will be printed, then it is moving, plus plus rit means it is moving, correct. So, now again it is not equal to this printing 20. and then printing 30.

So, now it is printing to the first that means rit is equal to numbers dot rend. So, then it is coming out when it is coming out it is printing 30, 20 and 10. So, when you are running the code. you should get 30, 20 and 10, right. So, in the C++ compiler, let us run this code.

You call it as reverse iterators. Yes, you can see 30, 20 and 10. So, what we can do, we will put one more element, let us say 40, 30 comma 40, for our understanding, right. So, now when I run and execute the code, so I will get you can see 40, 30, 20 and 10. So, from the reverse it is going all right.

So, initially the pointer will point to 40. So, not at the end all right. So, then it is printing 40 moving to 30. Location, 30 it is printing moving to 20, 20 it is printing moving to 10 all right. So, 10 will also be printed and then it is coming out of the loop.

So, in the next lecture, what we will do is talk about these specialized iterators. So, these specialized iterators. So, we are going to see stream iterators and insert iterators,

right? So, these two iterators we will see in the next lecture, right? So, now in this lecture, we have seen up to iterators and reverse iterators.

Thank you very much.