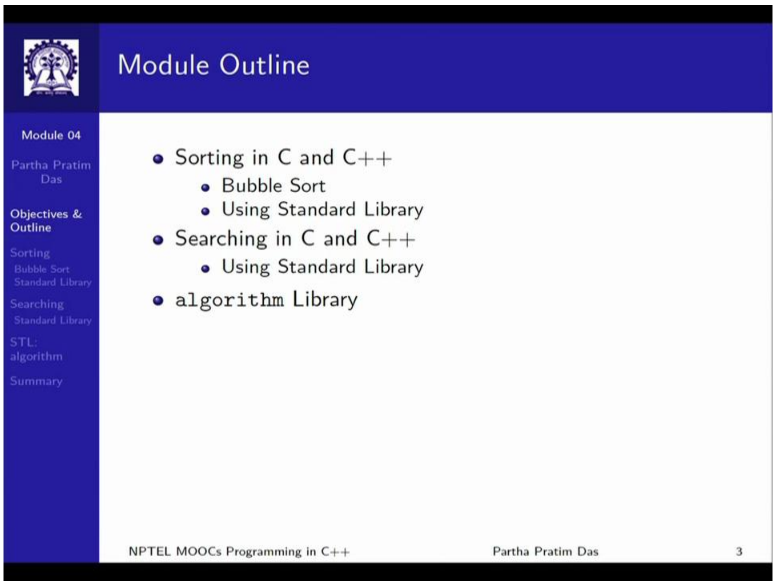
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 06
Sorting and Searching

Welcome to Module 4 of Programming in C++. In this module we are discussing sorting and searching. So, the objective is to look in to the implementation and use of sorting and searching in C and contrast them with C++.

(Refer Slide Time: 00:41)



Module Outline

- Sorting in C and C++
  - Bubble Sort
  - Using Standard Library
- Searching in C and C++
  - Using Standard Library
- `algorithm` Library

Module 04

Partha Pratim Das

Objectives & Outline

Sorting

Bubble Sort

Standard Library

Searching

Standard Library

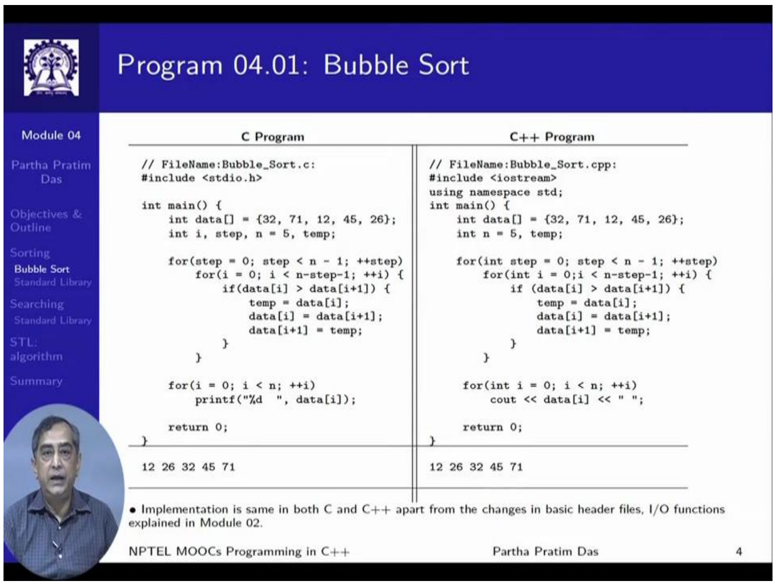
STL: algorithm

Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das 3

These are the topics that we are going to discuss.

(Refer Slide Time: 00:44).



Program 04.01: Bubble Sort

Module 04

Partha Pratim Das

Objectives & Outline

Sorting

Bubble Sort

Standard Library

Searching

Standard Library

STL: algorithm

Summary

C Program

```
// FileName:Bubble_Sort.c:
#include <stdio.h>

int main() {
    int data[] = {32, 71, 12, 45, 26};
    int i, step, n = 5, temp;

    for(step = 0; step < n - 1; ++step)
        for(i = 0; i < n-step-1; ++i) {
            if(data[i] > data[i+1]) {
                temp = data[i];
                data[i] = data[i+1];
                data[i+1] = temp;
            }
        }

    for(i = 0; i < n; ++i)
        printf("%d  ", data[i]);

    return 0;
}
```

```
12 26 32 45 71
```

C++ Program

```
// FileName:Bubble_Sort.cpp:
#include <iostream>
using namespace std;
int main() {
    int data[] = {32, 71, 12, 45, 26};
    int n = 5, temp;

    for(int step = 0; step < n - 1; ++step)
        for(int i = 0;i < n-step-1; ++i) {
            if (data[i] > data[i+1]) {
                temp = data[i];
                data[i] = data[i+1];
                data[i+1] = temp;
            }
        }

     for(int i = 0; i < n; ++i)
         cout << data[i] << " ";

     return 0;
}
```

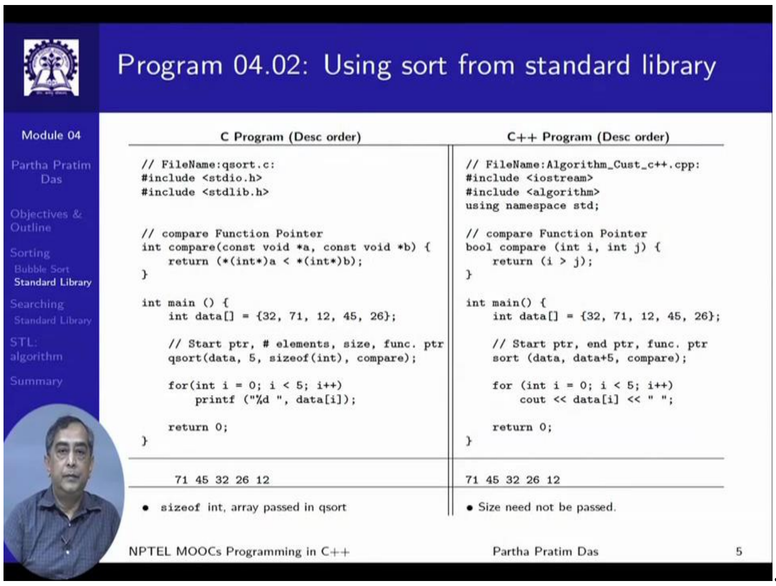
```
12 26 32 45 71
```

- Implementation is same in both C and C++ apart from the changes in basic header files, I/O functions explained in Module 02.

NPTEL MOOCs Programming in C++ Partha Pratim Das 4

So to get started, all of you know sorting, and you must have written several programmes in C to sort a set of integers given in an array. So here, we just show one of the most common sorting algorithms known as the bubble sort. You may or may not have used a bubble sort; if you have done some other selection or insertion sort, it is perfectly fine. We are not going to get into the logic of how the sorting is done. All that I want to show is between the two columns; left is the C programme, right is the C++ programme. Both are trying to do bubble sort using exactly the same strategy, the same algorithm; and, you can see the only difference they have between them, is the use of the IO header and the std name space. So far as the whole algorithm is concerned, the code is concerned, the arrays and comparisons are concerned, it is just identical. So, the first lesson that we take is, any sorting code that is written in C can be identically used in C++.

(Refer Slide Time: 01:57)



Program 04.02: Using sort from standard library

Module 04

Partha Pratim Das

Objectives & Outline

Sorting

Bubble Sort

Standard Library

Searching

Standard Library

STL: algorithm

Summary

C Program (Desc order)

```
// FileName:qsort.c:
#include <stdio.h>
#include <stdlib.h>


// compare Function Pointer
int compare(const void *a, const void *b) {
    return (*(int*)a < *(int*)b);
}

int main () {
    int data[] = {32, 71, 12, 45, 26};

    // Start ptr, # elements, size, func. ptr
    qsort(data, 5, sizeof(int), compare);

    for(int i = 0; i < 5; i++)
        printf ("%d ", data[i]);

    return 0;
}
```

```
71 45 32 26 12
```

- sizeof int, array passed in qsort

C++ Program (Desc order)

```
// FileName:Algorithm_Cust_c++.cpp:
#include <iostream>
#include <algorithm>
using namespace std;

// compare Function Pointer
bool compare (int i, int j) {
    return (i > j);
}

int main() {
    int data[] = {32, 71, 12, 45, 26};

    // Start ptr, end ptr, func. ptr
    sort (data, data+5, compare);

    for (int i = 0; i < 5; i++)
        cout << data[i] << " ";

    return 0;
}
```

```
71 45 32 26 12
```

- Size need not be passed.

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

Now, I would like to illustrate that, if I need to sort a set of numbers or set of strings, do I really every time write a sorting programme myself? I do that when I am trying to learn algorithms or when I am trying to learn the language, but subsequently if I want to sort a set of numbers I would not like to write the algorithm myself; rather, I will again go back to the standard library and try to use what standard library provides. Now, let us concentrate on the left column, which is the C programme. It is using a sorting function provided in the C standard library, in the header Std Lib dot h. This function is known as Q sort. The name certainly refers to the fact that this function uses quick sort as the underlying sorting algorithm.

Now, please concentrate on how the Q sort function is called. You can see that, there are 4 parameters. So let us see, what are the things that you need to tell the Q sort programme, so that it can sort your data. Certainly, you need to tell where the data exists. So, that is the container, which is an array data here. So, that is the first parameter. Now, the array would have any number of items, any number of elements; how will you sort, know how many elements does it have or out of the containing elements, how many of them you want to get sorted. So, the second parameter tells Q sort that, how many of the array elements you want to get sorted. It means that, you always start with an index 0 and take as many elements as specified in the second parameter. So, here the second parameter is 5, which means that Q sort should sort data array from index 0 to index 4, which in this case happens to be the whole array.

The third parameter is little tricky, and right now you may not understand why the third parameter is required. The third parameter says that please provide the size of every element in bytes. Now, this is an integer array, which means that every element is an int. So, the size of every element can be computed from using the size of operator in C, so you specify size of int which for a 32 bit system will pass 4, since int, typically in 32 bit is represented by 4 bytes as the third parameter. May be difficult for you to understand exactly what we need this, but the basic logic is that, since Q sort does not know what type of data your array contains, it could contain int type of elements; it could contain char type of elements; it could contain pointed type of elements; it would contain structure type of elements; it is not possible for Q sort to know after it has got the address of the zeroth indexed element where should it find the first indexed element. So, if it knows the size of the element it will be able to add that as an offset to the starting address of the array to get the first element. Then it can again add that same size of int offset to the first element's position to get the address of the second element and so on. So, that is the implementation for which you need this information. So, that is the third parameter.

The fourth parameter is the most interesting. The fourth parameter is a function pointer. In module 1, you will remember we have talked about function pointers in depth. We have recapped that and if you just move your attention to the top, between the includes and main you find that a function pointer compare has been defined, implemented for the purpose of use by Q sort. Now, what is the role of this compare function? This compare function will take 2 values compare them and say which one the first one is smaller or the second one is smaller. So here, we are shown a compare function, where we are using less than as a comparison. If the first one is smaller than the second one, it will return it true and otherwise it will return a false. This is the basic purpose of the compare function.

Now you will wonder why do I need to do such a complicated thing as writing a separate function pointer function and passing that function pointer. You will please have to understand that, when Q sort was written in the library or if you want to write a Q sort kind of function now and you are not told what kind of elements we will need to sort, how will you know how to compare those elements? For example, int can be compared very easily, but if I have say instead of int, if I now give you an array of strings. Now, the array of strings cannot be compared simply by writing a less than or greater than in C. We are not talking about C++ string right now, which you already know has a comparison operator. So, in C if I have to compare strings my values are actually pointed to character, but my comparison has to happen to the use of some STRCMP function. Now how would Q sort know, Q sort did not know what kind of data you will give Q sort to compare them, therefore Q sort also does not know how to compare 2 such data items. And it is your responsibility to provide that compare function to Q sort.

So you need to have this fourth parameter, which is the compare function pointer. If you look into the header, the signature of the compare function pointer you see something which is even more disturbing. You can see that the parameters are declared as of const void star that is there const pointers to constant data of unknown type, why is it that? Because again, Q sort could not have put a signature for the compare function pointer, because it does not know the type of data items. So, all of these assuming that you have a pointer to some type which is not known and, since you do not know the type while you actually want to do the comparison in the return statement of the compare function, you will have to first tell the compiler that, what indeed I now have is an integer. So, you cast the void star pointer first to int pointer, then you de-reference that int pointer to get the integer value and then you compare the two integer values.

So, so much need to happen for the Q sort to be used. And that is one of the reasons possibly that even though Q sort is available in the C standard library as a default sorting algorithm, the use of Q sort has not been that popular in the C community. Often, people prefer to write their own functions; bubble sort, selection sort or insertion sort, merge sort, whatever, and given the data type of elements that they want to sort and just use that.

Let us now look at the right side of the column. The C++ way of doing it, I will not get into the differences in terms of IO's you already know that. What you must focus in terms of include is, we include another C++ standard library header called algorithm, which is very very interesting. Algorithm is a header, which contains a whole lot of algorithm codes that you need to often use in your programmes. So, algorithm contains one of the components is the sort algorithm. So, what you call Q sort in C, in C++ this component in algorithm is called sort.

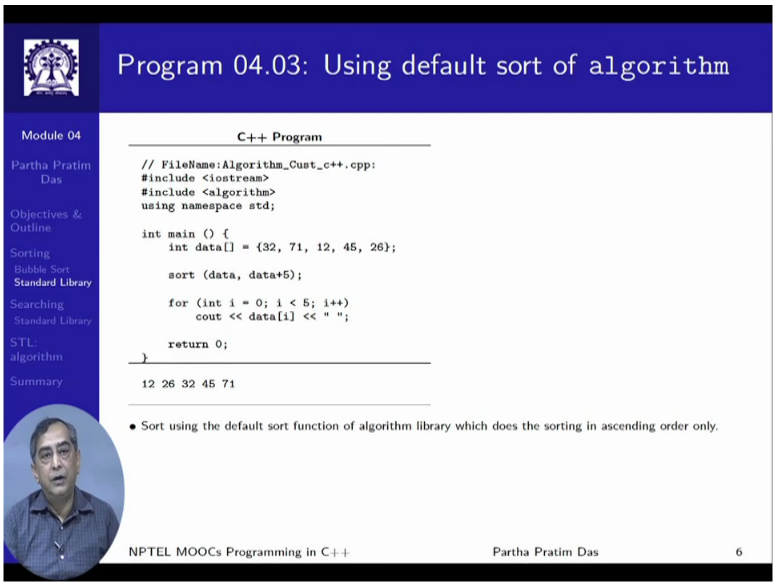
We are again sorting the same data; we are again sorting in the same order. What I need to specify for sort, let us look at the parameters. Certainly, the container which is same as what I did in Q sort. The second I need to tell, how many elements from the beginning, here this we specify in a little bit different way. In Q sort, we just specified the number of elements; here we pass the pointer to the first element, after point of attention. So, we pass data plus 5 which means that we pass the pointer to the index 5 elements, which is just beyond the part of the array that we want to sort. Which in other words, means that you sort 5 elements starting from data? We will look at these details later on, this is called a range in C++.

The third parameter as of Q sort is not required, because as you have seen before also, that C++ compiler can deduce things based on the type. So type, it knows that data is of int type array. So, the elements will be of int types. So, you do not need to separately specify the size of int as a parameter. The third parameter of sort is same as the fourth parameter of Q sort that you need to give the comparison algorithm. This certainly cannot be deduced, because depending on which type of data you are sorting, your comparison strategy would be different. But, what is significant is, if you again move your attention back to where this compare function is defined in the C++ part.

Now you do not have that complication that you had in C, of using parameters which were of const void star type or the expression which does a whole lot of casting and de-referencing. Now, you can declare the parameters exactly as you declare in case of a normal function, int i, int j and in the return you simply compare the, whatever that comparison logic is we will just write that here. It is pretty much like a normal comparison function and that function pointer; function name will be passed as a third parameter. You can do this in C++, because of several features that C++ introduces which we will slowly introduce to you. So, you will understand why this was not possible in C, but it is possible in C++ to write the compare function with any type that the sort function did not know beforehand, but can still be able to absorb that.

So, with this certainly, it becomes a lot more elegant and efficient to use the sort programme from the standard library. And the consequence therefore is that, the C++ programmers really write their own sorting programmes, they just make use of the sort as is given here. There are small nuances to note; the convention of the direction of sorting, whether it is decreasing order or increasing order differs between the Q sort implementation and the sort implementation. So, you will if you again focus on the two compare functions that we have written, you will find that in C we have used a less than to be true, in C++ we have given greater than to be true because, in both cases we want the sorting to be done in descending order. So, this is just a matter of convention that exists in C++ sort.

(Refer Slide Time: 14:53)



Program 04.03: Using default sort of algorithm

Module 04

Partha Pratim Das

Objectives & Outline

Sorting

Bubble Sort

Standard Library

Searching

Standard Library

STL: algorithm

Summary

C++ Program

```
// FileName:Algorithm_Cust_c++.cpp:
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
    int data[] = {32, 71, 12, 45, 26};

    sort (data, data+5);

    for (int i = 0; i < 5; i++)
        cout << data[i] << " ";

    return 0;
}
```

```
12 26 32 45 71
```

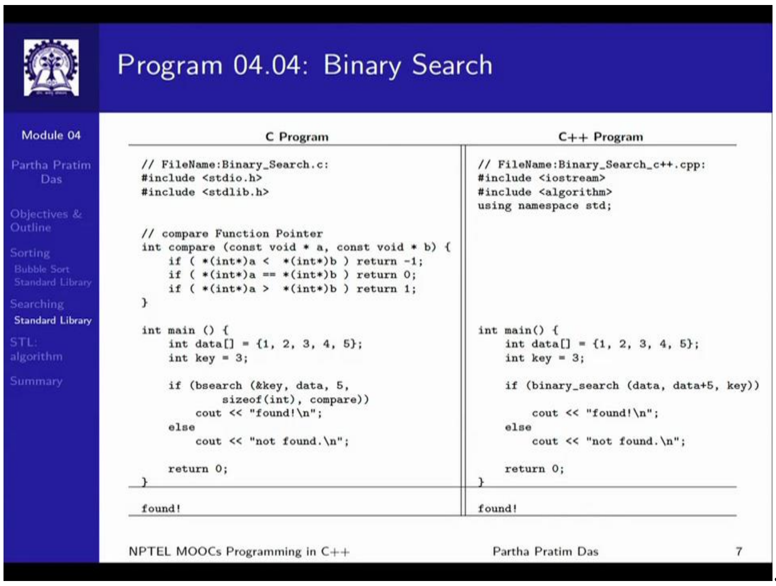
- Sort using the default sort function of algorithm library which does the sorting in ascending order only.

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

This is just another version of the sort. Here, if you look into the, particularly the call of the sort function, you see that, the first two parameters are there, but the third parameter is not there that is, the compare function is not provided here. This is a short cut that C++ standard library allows you; is if you are trying to sort arrays of type which is known to the C++, that is of the built in types then it is optional to give the comparison function.

The comparison function is not necessary to be provided and if you do not provide that, then by default the sorting happens in the ascending order. If you still want to sort this array in descending order, you will again need to provide the compare function because your direction of comparison, that is whether the first parameter being greater than the second is true or otherwise, will change between the ascending sorting and the descending sorting.

(Refer Slide Time: 16:08)



Program 04.04: Binary Search

Module 04

Partha Pratim Das

Objectives & Outline

Sorting
- Bubble Sort
- Standard Library

Searching
- Standard Library

STL: algorithm

Summary

C Program

```
// FileName:Binary_Search.c:
#include <stdio.h>
#include <stdlib.h>


// compare Function Pointer
int compare (const void * a, const void * b) {
    if ( *(int*)a <  *(int*)b ) return -1;
    if ( *(int*)a == *(int*)b ) return 0;
    if ( *(int*)a >  *(int*)b ) return 1;
}

int main () {
    int data[] = {1, 2, 3, 4, 5};
    int key = 3;

    if (bsearch (&key, data, 5,
            sizeof(int), compare))
        cout << "found!\n";
    else
        cout << "not found.\n";

    return 0;
}
```

```
found!
```

C++ Program

```
// FileName:Binary_Search_c++.cpp:
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int data[] = {1, 2, 3, 4, 5};
    int key = 3;

    if (binary_search (data, data+5, key))

        cout << "found!\n";
    else
        cout << "not found.\n";

    return 0;
}
```

```
found!
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

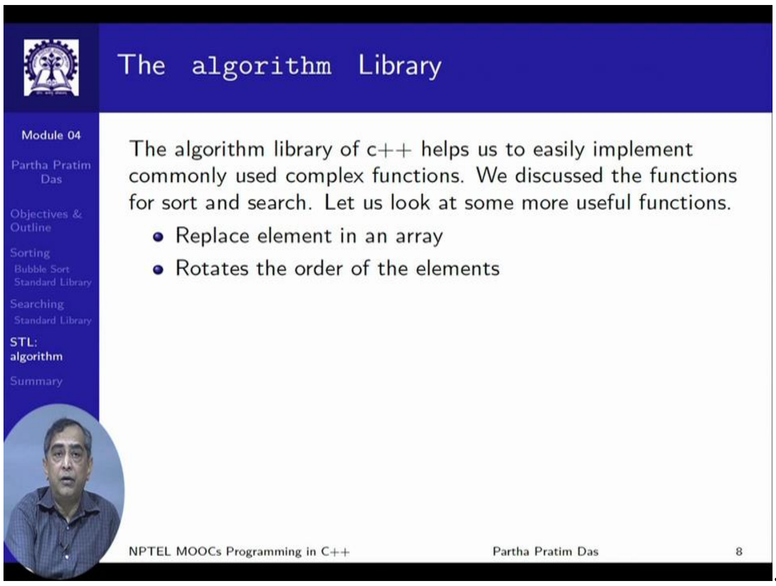
Next, we will move on to Binary Search. We all know that, a binary search is a very frequently used algorithm in programming, which given an array of elements, which are already sorted. Binary search can find out whether a given key exists in that array, if it does then it can also say, what is the position of that element in the array? This is, of the simple searching mechanisms, this is one of the most efficient one. And, both C and C++ standard library have mechanisms to do binary search. In C, it again is available from Std Lib dot h. It is a function called b search. In C++, it is again available from component algorithm and it is called binary underscore search. If we will look into how this is used in C look on the left hand side, by b search the first parameter is a key. The key has to be passed as its address, so you pass ampersand key.

There is a reason why you need to do this, because again, you do not know the type of elements that you have in the array, and therefore the type of the key that you want to search. So, you do not know the type of the variable key in general. So, you cannot write b search with that so you again make use of the fact that you can use a pointer to int and consider that to be a pointer to void. So, b search uses the first parameter which is void star. And that is the reason you need to give it the address. The second parameter is, the array in which you want to search, certainly this has to be sorted. The third parameter is the number of elements. This is pretty much like Q sort. The fourth parameter is the size of every element and the fifth parameter is the function pointer of the comparison function.

Point to note is, binary search has to make a three way decision, unlike sorting where you just need to know whether it is less or is not less, but binary search has to when it is looking at a particular element in an array, it has to deal with 3 possibilities. One, is the element that it is looking at may actually be equal to the key, actually may be the key so then you are done with the search. So, you do not need to do anything you just return that index value of the position of the array. Second, it could be less than that. If you have your array sorted in the increasing order and your value is less than the element that you are looking at, your key is less than the element that you are looking at, you know that you have to look at the left part of the array. And in the third case, you have to look at the right part of the array. So, your comparison function has to be a three way function, pretty much like the STRCMP function we have in the string dot h standard library in C. So on top here, I show how you can write such a compare function for the case that we are dealing with which returns minus 1, 0 or plus 1 as the case may be. And, all the nuances of using parameters as void star pointer and casting them, dereferencing them, as we did it the case of Q sort, will also be involved in this case as well.

Let us look at the C++ form for this. The binary underscore search function from algorithm component takes the first parameter as the container, which is the data, the second parameter is the end of the range. This is pretty much like sort as we have seen. And it takes the third parameter, which is key. And here, I am showing an example, where the compare function is not explicitly provided, because as I have already discussed in case of sort that, if you are doing a search for elements of built in type the compiler already knows how the comparison is done. So, you do not need to explicitly put a function pointer for comparison. And, with this, the binary search will happen. If you compare these two you will find, the ease of using the binary search in C++ is immensely more compared to the ease of using it in C. And again, for that matter binary search in C++ is very often used and nobody will write a programme for doing a binary search for any type of data container that someone has.

(Refer Slide Time: 21:24)



The algorithm Library

Module 04

Partha Pratim Das

Objectives & Outline

Sorting
Bubble Sort
Standard Library

Searching
Standard Library

STL: algorithm

Summary

The algorithm library of c++ helps us to easily implement commonly used complex functions. We discussed the functions for sort and search. Let us look at some more useful functions.

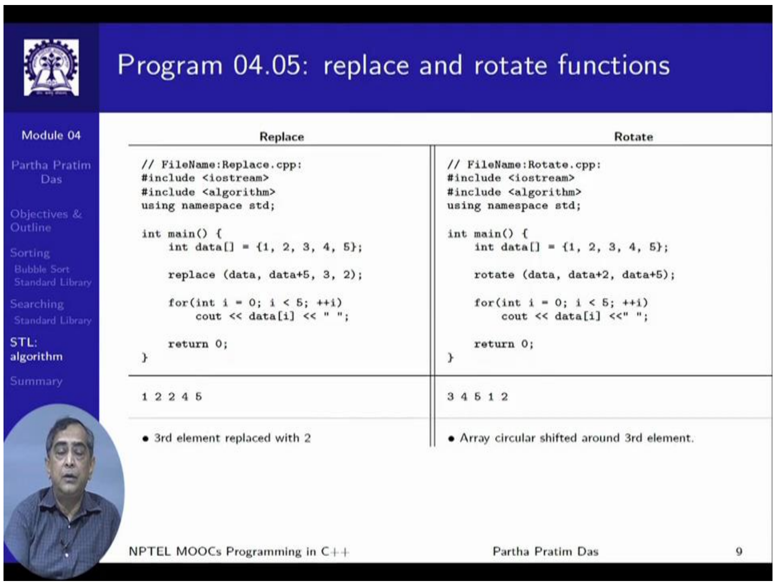
- Replace element in an array
- Rotates the order of the elements

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

.

Specifically, in this algorithm library which is a very interesting library, because what it is saying that, if you are using C++, we give you a set of whole set of common algorithm. So, we just saw how sort and search can be used from algorithm library. There are several others, like replacing elements in an array, rotating the order of elements.

(Refer Slide Time: 21:48).



Program 04.05: replace and rotate functions

Module 04

Partha Pratim Das

Objectives & Outline

Sorting

Bubble Sort

Standard Library

Searching

Standard Library

STL: algorithm

Summary

Replace	Rotate
see code below	see code below

Replace:

```
// FileName:Replace.cpp:
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int data[] = {1, 2, 3, 4, 5};

    replace (data, data+5, 3, 2);

    for(int i = 0; i < 5; ++i)
        cout << data[i] << " ";

    return 0;
}
```

```
1 2 2 4 5
```

- 3rd element replaced with 2

Rotate:

```
// FileName:Rotate.cpp:
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int data[] = {1, 2, 3, 4, 5};

    rotate (data, data+2, data+5);

    for(int i = 0; i < 5; ++i)
        cout << data[i] <<" ";

    return 0;
}
```

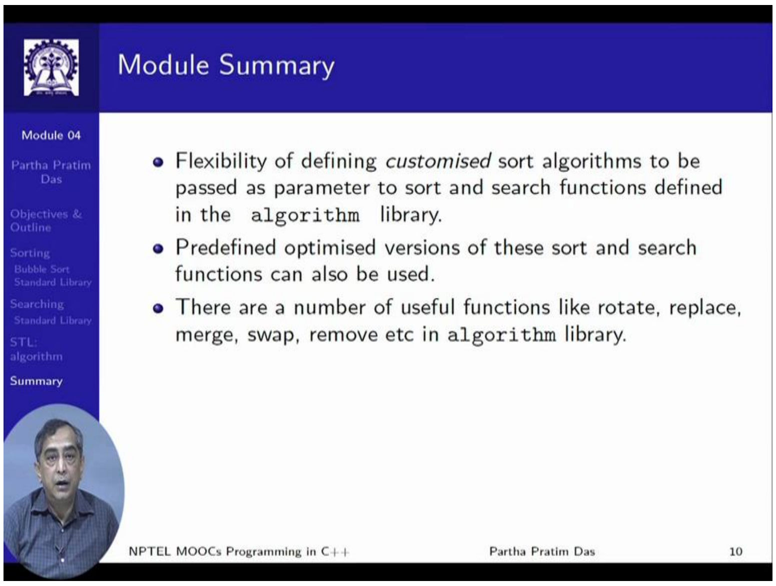
```
3 4 5 1 2
```

- Array circular shifted around 3rd element.

NPTEL MOOCs Programming in C++ Partha Pratim Das 9

So, these are examples of replace, rotate codes. We will not go through step by step through them. By now, you should be familiar to use to understand this. Please look up the manual or the book, for the details on these algorithms and start using them. You will really find that, writing things in C++ using the algorithm library becomes very very easy, because most of the common algorithms are already available and very easy to use.

(Refer Slide Time: 22:20).



Module Summary

Module 04

Partha Pratim Das

Objectives & Outline

Sorting

Bubble Sort

Standard Library

Searching

Standard Library

STL: algorithm

Summary

- Flexibility of defining *customised* sort algorithms to be passed as parameter to sort and search functions defined in the `algorithm` library.
- Predefined optimised versions of these sort and search functions can also be used.
- There are a number of useful functions like rotate, replace, merge, swap, remove etc in `algorithm` library.

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

In this module, we have shown that how basic sorting and searching can be done in C++ with a much better ease and efficiency and we particularly illustrate that, unlike C where we often end up writing our own sorting code and searching code, in C++ there would be hardly be any reason to do so. No matter what kind of container and data we are using we will show that this is also true for several other algorithms that we need to use, including merge, swap, remove, all of these, different ones. And, I would encourage you to study these algorithm components and start using them. And then the beauty of the whole thing is, you really do not need to know a lot of C++ to be able to use them, because their use their design and the way they are organised are quite intuitive and you can just study from the manual and start using them.