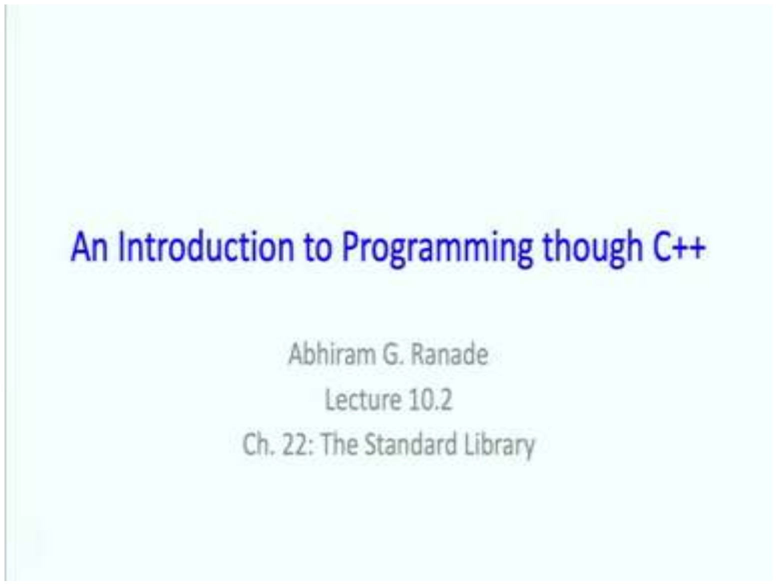
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture-23 Part-01
The Standard Library
Class string

(Refer Slide Time: 0:18)

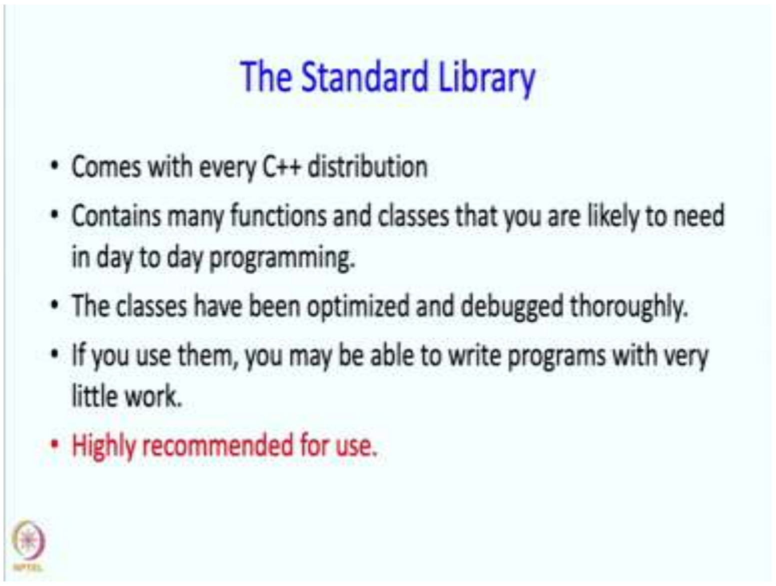


An Introduction to Programming though C++

Abhiram G. Ranade
Lecture 10.2
Ch. 22: The Standard Library

Hello and welcome to the NPTEL course on ‘An Introduction to programming on C++’. I am Abhiram Ranade and today’s lecture is on standard library of C++, this is discussed in Chapter number 22 of the text.

(Refer Slide Time: 0:38)

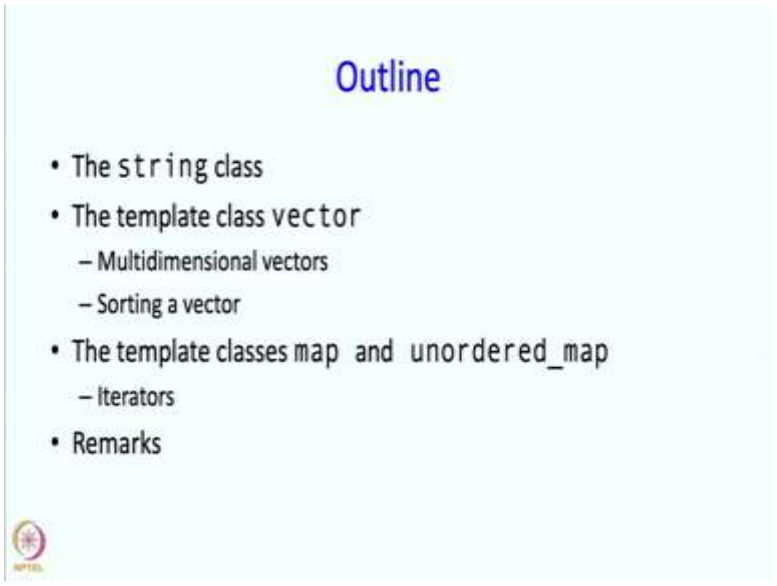


The Standard Library

- Comes with every C++ distribution
- Contains many functions and classes that you are likely to need in day to day programming.
- The classes have been optimized and debugged thoroughly.
- If you use them, you may be able to write programs with very little work.
- Highly recommended for use.

So, the standard library comes with every C++ distribution and it contains many functions and classes that you are likely to need in day to day programming. These classes have been optimized and debugged thoroughly. So, if you use them, they will be reliable and you may be able to write programs with every little work. So, definitely use them if the need arises. Definitely do not invent your own classes, but learn these classes and use them.

(Refer Slide Time: 1:10)



Outline

- The `string` class
- The template class `vector`
  - Multidimensional vectors
  - Sorting a vector
- The template classes `map` and `unordered_map`
  - Iterators
- Remarks

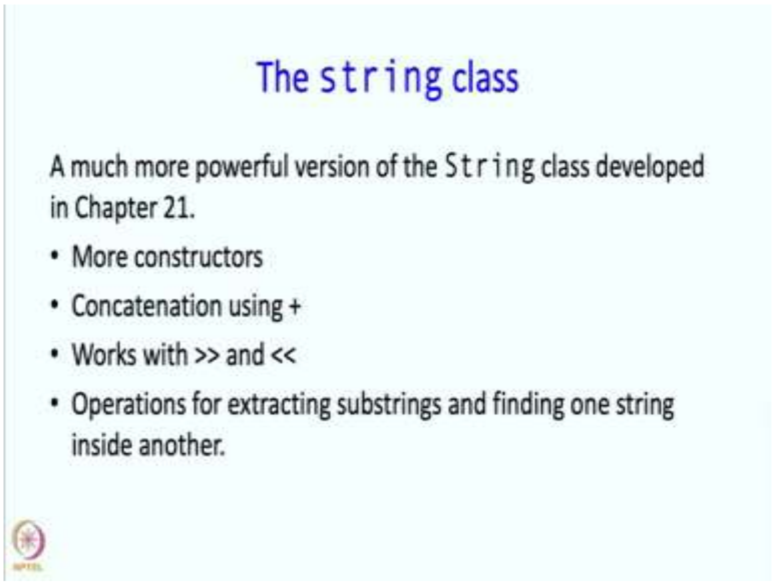
So, here is what we are going to do in this lecture. So, I am going to talk about the string class. In the last lecture we talked about a class, which we ourselves designed and which we called String.

So this is going to be string but with everything lower case. In some sense this class is implemented in a manner similar to our String class and in fact all the classes that I am going to talk about, will have implementations which will need the heap memory.

Okay, so we will deallocate and allocate heap memory but yet all of those things will happen behind the scenes and to that extent the implementations will be similar to capital String class implementations which we had had. But, of course, these classes are going to be something different and there will be differences as well and, in this course, as well we are not going to talk about exactly how these classes are implemented but just to get a sense that how they might be implemented.

Okay, we have discussed the capital String class, so we will have some understanding, all of this will not seem like too much magic to you. So, we will talk about string class, we'll talk about a class called a vector class. And we will talk about classes a map, and unordered map and then we will conclude. So, the string class from the library, okay.

(Refer Slide Time: 2:46)



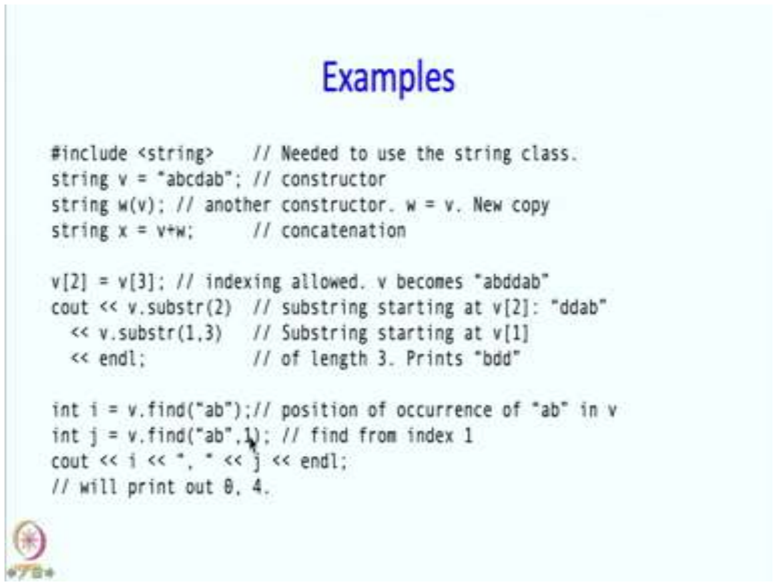
The `string` class

A much more powerful version of the `String` class developed in Chapter 21.

- More constructors
- Concatenation using +
- Works with >> and <<
- Operations for extracting substrings and finding one string inside another.

So, this is a much more powerful version of the string class we developed in chapter 21. There more constructors, concatenation using +   and you can directly print out or read into the strings. You do not have to go through, go through null terminated arrays or anything like that. And there are operations for extracting substrings or finding one string inside another.

(Refer Slide Time: 3:15)



Examples

```
#include <string>    // Needed to use the string class.
string v = "abcdab"; // constructor
string w(v); // another constructor. w = v. New copy
string x = v+w;      // concatenation

v[2] = v[3]; // indexing allowed. v becomes "abddab"
cout << v.substr(2)  // substring starting at v[2]: "ddab"
  << v.substr(1,3)   // Substring starting at v[1]
  << endl;           // of length 3. Prints "bdd"

int i = v.find("ab");// position of occurrence of "ab" in v
int j = v.find("ab",1); // find from index 1
cout << i << ", " << j << endl;
// will print out 0, 4.
```

So, we are going to see some examples. This is not going to be a comprehensive discussion of the string class, for that I will recommend that you look at the online documentation. So, to use this class you need to include the header file string. So, here is how you might use it. So, I might create a string and while creating I might initialize it to this string constant “abcdab”. Now, I can write string w and in parenthesis I can write a string from which you are going to copy that string.

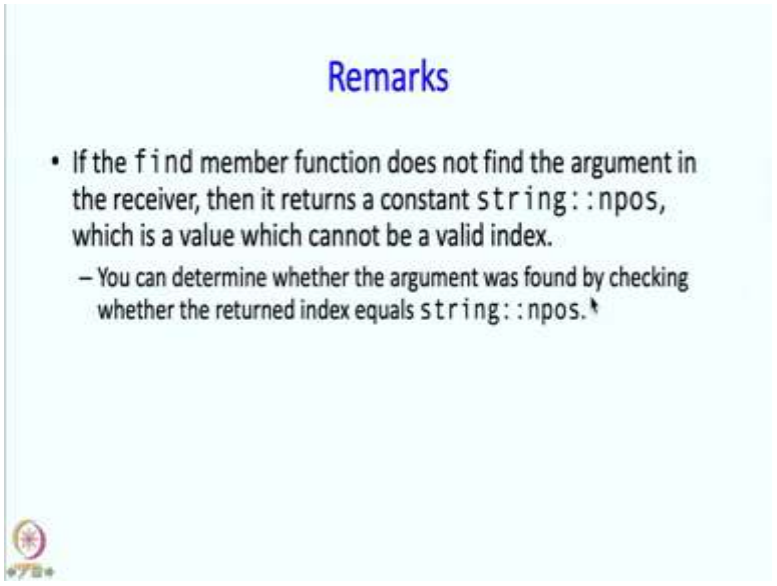
So, this is another constructor, but this copies directly. So, that v is directly copied, so I could have written this to string w=v as well, but this is another style of writing initial values. I can write string x=v+w, so that it is just concatenation and the concatenation is put into string x. I can index and index operations can be on both sides, so in this particular case v2 is changing to v3 so we had abcdab before so that will now become abddab.

Here is an interesting operation, we are doing cout<<v.substring there should be a semicolon over here, but anyway, so that the substring 2 means the substring of v starting at 2. So, this is v, so this 012, 012 but remember we just changed this c to d. So, the substring of v starting at 2 will be ddab. So, v, substring (1.3) says that I want a substring which starts at 1 but whose length is 3. So, substring of v which starts at 1, so v is now this, so 0 1, so it starts at b but its length is 3, so this will be bdd.

So, for this part we will be end up printing bdd, okay. Then we can find, we can look for strings inside a given string. So, for example, I can write v.find, so this is for v and we are finding ab inside it. So, where does ab appears, it appears at 0, so this will return 0. But suppose I execute v from position 1.

So, this says find ab but execute it from position 1. So, this is our current v, I am not going to start from here, I am going to start from here. So, in this case, in this case my occurrence is going to be at 01234, so this should be at occurrence 4. So, this will end up printing 0 and 4 because i will be set to 0 j will be set to 4.

(Refer Slide Time: 6:39)

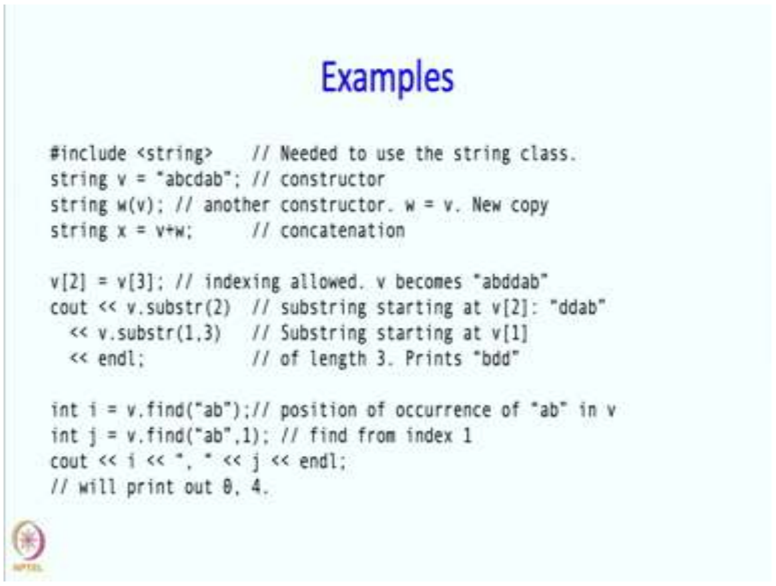


Remarks

- If the `find` member function does not find the argument in the receiver, then it returns a constant `string::npos`, which is a value which cannot be a valid index.
  - You can determine whether the argument was found by checking whether the returned index equals `string::npos`.

So the find function needs an explanation, because if the find member function does not find the argument in the receiver, then it returns a constant string::npos. So which is not a valid index. So, what you can do, you can check whether the returned index equals string::npos. So that we know that the string is not present.

(Refer Slide Time: 7:10)



Examples

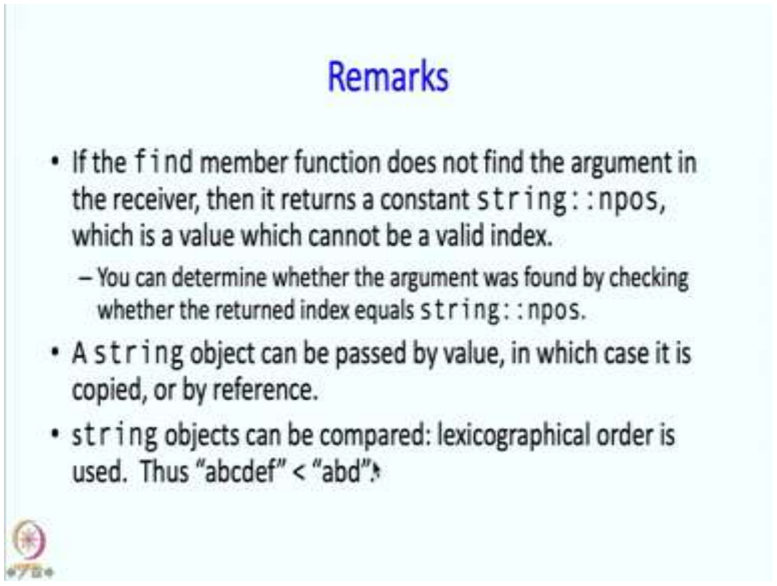
```
#include <string>    // Needed to use the string class.
string v = "abcdab"; // constructor
string w(v); // another constructor. w = v. New copy
string x = v+w;      // concatenation

v[2] = v[3]; // indexing allowed. v becomes "abddab"
cout << v.substr(2)  // substring starting at v[2]: "ddab"
  << v.substr(1,3)   // Substring starting at v[1]
  << endl;           // of length 3. Prints "bdd"

int i = v.find("ab");// position of occurrence of "ab" in v
int j = v.find("ab",1); // find from index 1
cout << i << ", " << j << endl;
// will print out 0, 4.
```

So, going back over here, if we were looking for say x y, then I would have been set to string::npos, okay. So npos probably is some negative value, for example, so which cannot be the position of any occurrence, so that is how you can actually check that whether the string appears or not appears.

(Refer Slide Time: 7:35)

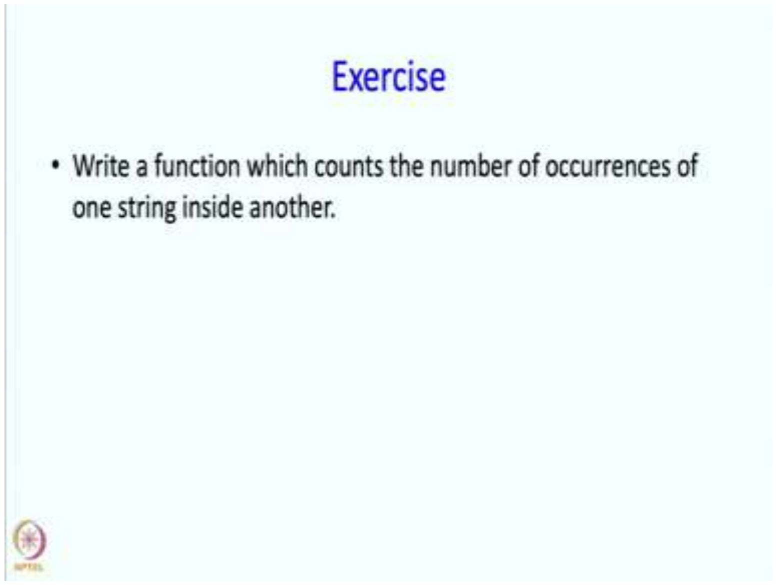


Remarks

- If the `find` member function does not find the argument in the receiver, then it returns a constant `string::npos`, which is a value which cannot be a valid index.
  - You can determine whether the argument was found by checking whether the returned index equals `string::npos`.
- A `string` object can be passed by value, in which case it is copied, or by reference.
- `string` objects can be compared: lexicographical order is used. Thus "abcdef" < "abd".

Okay, so, a string object can be passed by value, in which case it is copied, or by reference. So, if you are passing strings to a function, this is how you can do both. But, of course, usual rules, usual rules apply. String objects can also be compared and for this, the lexicographical order about which we talked about some time ago is used, the dictionary order. So, just to clarify abcdef appears earlier in the dictionary, so this will be deemed smaller than this.

(Refer Slide Time: 8:15)

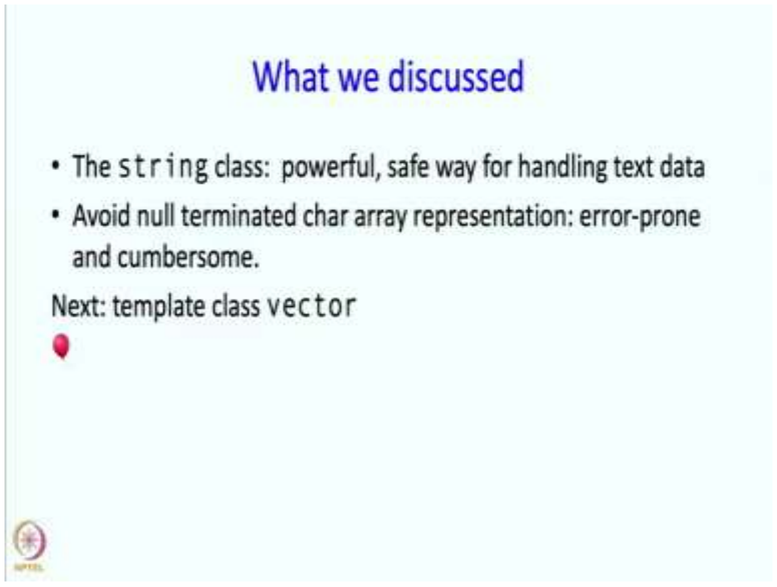


Exercise

- Write a function which counts the number of occurrences of one string inside another.

So here is an exercise for you. Write a function which counts the number of occurrences of one string inside another. So, you know, in our old string when there are two occurrences so your function should compute, you should print out both the occurrences. So, you will have to use the find command and slightly in slightly imaginative manner.

(Refer Slide Time: 8:38)



What we discussed

- The `string` class: powerful, safe way for handling text data
- Avoid null terminated char array representation: error-prone and cumbersome.

Next: template class `vector`

Okay, Alright so what have we discussed, we have discussed some features of the string class which is a powerful safe way for handling text data. So, you really should try to use the string class as much as possible and avoid null terminated char arrays, because those are error-prone and also cumbersome. In the next segment, we are going to talk about a class called vector class, but before we do that let us take a short break.