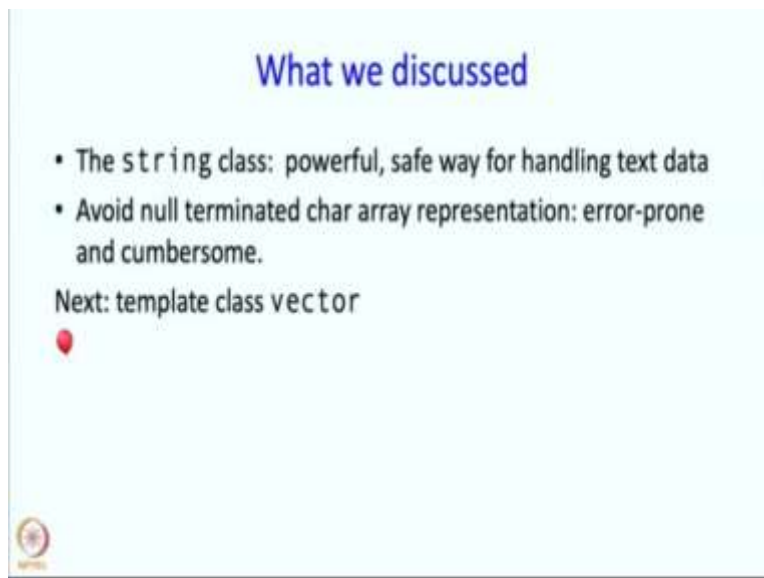


**An Introduction to Programming through C++**  
**Professor Abhiram G. Ranade**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Bombay**  
**Lecture-23 Part-02**  
**The Standard Library**  
**Class vector**

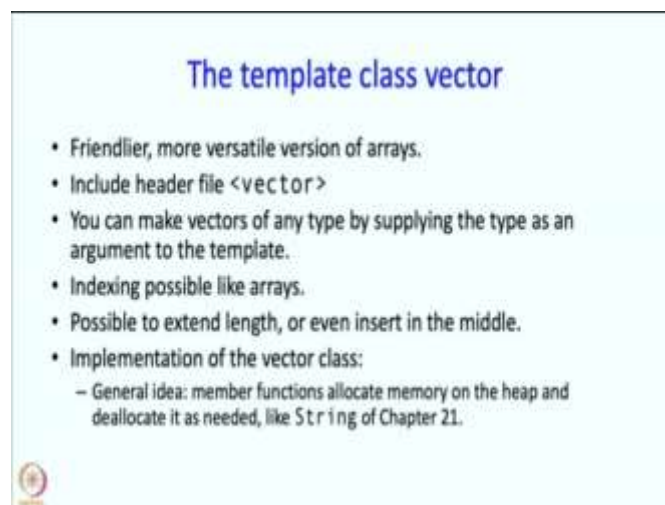
Welcome back!

(Refer Slide Time: 0:19)



In the previous segment we discussed this string class.

(Refer Slide Time: 0:25)

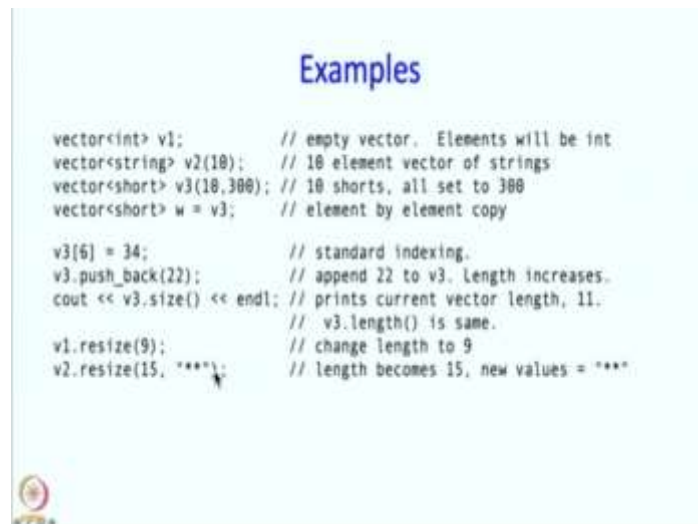


In this segment we will be discussing the vector class. So, the class vector is, should be thought of as a friendlier and a more versatile version of arrays. To use it you have to include the header file `vector`. You can make vectors of any type by supplying the type as an argument to the template and we will see this in a minute. So, like arrays indexing is possible. But, there are more things, you can extend the length and you can also insert in the middle of the arrays.

And the implementation of the vector class is sort of like the string class, in the sense that it will also allocate memory and memory from the heap. Okay. So there will be member classes but they will do the memory management and all that will happen automatically like the way it happened for the string, to the string class that we developed. But, of course, the functionality of

the vector class is different, so there will be differences also, but this whole idea of having the memory management happen behind the scenes is, of course, exactly like that.

(Refer Slide Time: 1:40)



So, I am going to discuss the string and the vector class also just by taking examples and again this is not going to illustrate every feature. To do that, you should look at documentation on the web. So, I can create a vector of integers by specifying int inside the angle braces or this is what is called the template argument. So, if I write `vector<int> v1`, I get an object called v1, but it is an empty vector.

So there are no elements yet but whenever elements come in, they will be of type int or you are expected to put in elements of type int. So, here is a vector of strings, you can have vector of anything, any type. So, here is a vector of strings and this time the name has an argument. So, this really is a constructor. So, if you have a one argument constructor, it creates, automatically creates a ten element vector of strings. The strings are initially empty, there is nothing. It is a 10 element vector of strings. Now, this has a more interesting constructor. First of all it is a vector of shorts. So, V3 is going to be a vector of shorts. It is going to have 10 elements, but if you have this additional argument, this additional argument says that the values of all the elements will be 300. So, v3 will store 10 copies of 300 once one per element .

I can write an assignment and I can, I can copy vectors directly just by specifying the entire name. So, to that extent vectors are like structures and, of course, you know that vectors are classes, so they are indeed, they are indeed structures. But they also have parts which are on the heap, so typically the data is all on the heap. They can do indexing so v3 is this thing we defined over here.

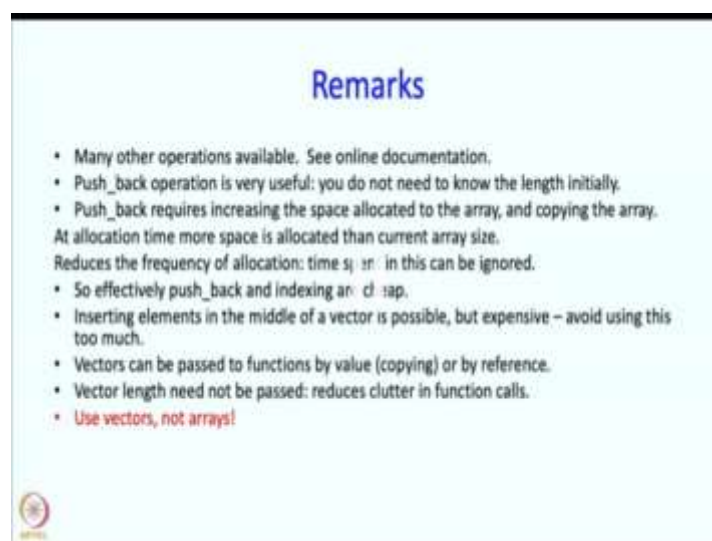
This vector that we defined over here and I can said the 6<sup>th</sup> element to be 34 and I can also look at any element inside that, whatever I want. Here is a more interesting operation. I can push back an element so this going to append 22 to v3. So, originally v3 had length 10 and it contained 10 300's, but now it is going to contain an extra element so it going to have elements 0 through 9 plus the 10<sup>th</sup> element.

And that 10<sup>th</sup> element is going to be 22. Or maybe this will require some memory allocation, unless you allocated a large amount of memory to begin with. But potentially this will require memory allocation and that will happen behind the scenes. You do not really need to even know about that.

I can find the size of a vector or I can also use length here if I wish. This will just print out the current length or current size and in this case 11 will get printed out. I can change the length. This says change the length to 9, so originally we had 0 length, so this is going to make the length to 9 .

This is going to change the length of v2. Remember v2 was a vector of length 10. All the elements were empty so this is going to have a length of 15. The new values will be initialized. They will be initialized to a string consisting of 2 stars. So there are lots of, there is a rich range of functions, member functions and constructors so you can do lots of things with very small command.

(Refer Slide Time: 5:43)



Yeah, as I said you should look at online documentation to get the whole range of commands or operations that are available. Now, push\_back is an interesting operation because often you have to guess the length of an array and you have to guess it and you have to allocate an array, but with the push back, you can just keep adding elements into your vector. So the guess work is gone. Yes push back will require increasing space after increasing the space the copying will also happen. So, it is going to be little more expensive than just assigning a value to any element. However, it is done very cleverly. So, it is done cleverly in the sense that suppose you exhaust the current allocation of space and you add a, you push back one more element, so what is done is a much larger space is given to you, not just that extra element. So that space is allocated and you copy your current vector into this new space.

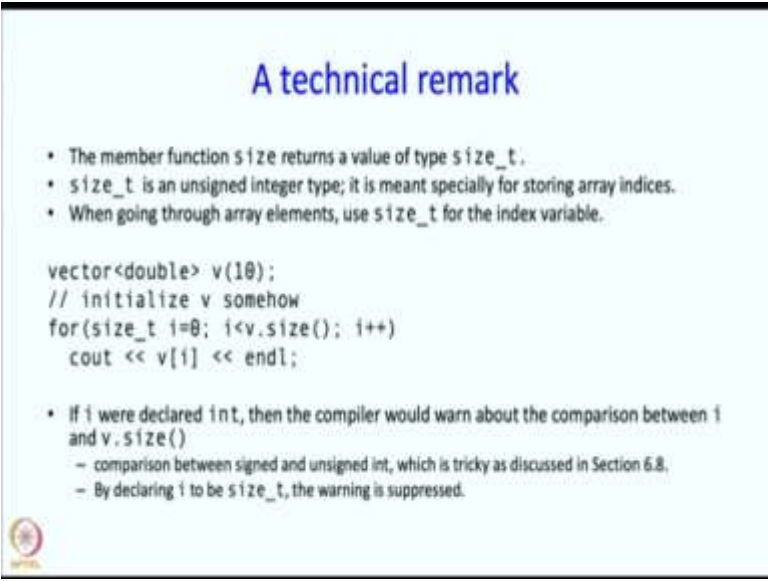
But because you get large space whenever you ask for it, your copying and reallocation operations do not happen that frequently. And there is a way to do it just right, so that really the effort you spend in reallocation over the entire course of the program turns out to be really quite negligible. So, effectively push back and indexing can be thought of as a cheap operations. So, this is one of the things which is really great with vectors that you can keep pushing back and it

is a cheap operation, so do not really worry about it. On the other hand, inserting elements in the middle of vector is possible but is an expensive operation.

And you should really not do this too commonly. I mean if you are doing this too commonly, you had better have really good reason so do not insert casually in the middle. Vectors can be passed to functions by value, or by reference and if you pass by value, things get copied. So, if you make a modification in the call function, then the vector in your calling function does not get modified but, of course, if you have passed it by reference, then the modification will be happening in your calling copy as well.

Now a nice thing about vector is that when you pass a vector either by reference or by value, you do not have to pass the length. So this just reduces the clutter. So your functions looks, your function calls look more compact. So, I would definitely advice you that now that you know about vectors, just start using them. Do not use arrays. Because they are just more elegant, more compact, more elegant, they can do lot more things.

(Refer Slide Time: 8:54)

A technical remark slide with a light blue background. The title "A technical remark" is in blue. It contains a bulleted list of three points, a code snippet, and another bulleted list. The code snippet shows a vector of doubles being initialized and iterated over. The second bulleted list explains a compiler warning about comparing a signed int with an unsigned int, and how declaring the index variable as size\_t suppresses the warning.

### A technical remark

- The member function `size` returns a value of type `size_t`.
- `size_t` is an unsigned integer type; it is meant specially for storing array indices.
- When going through array elements, use `size_t` for the index variable.

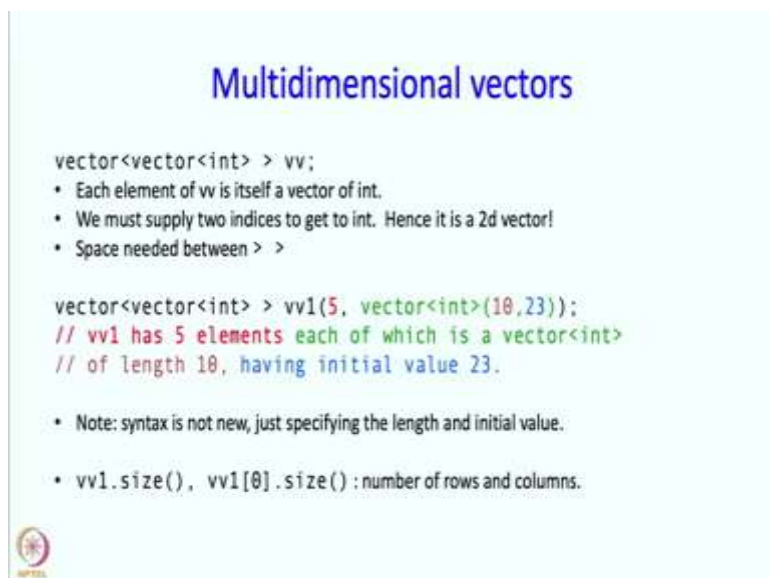
```
vector<double> v(10);  
// initialize v somehow  
for(size_t i=0; i<v.size(); i++)  
    cout << v[i] << endl;
```

- If `i` were declared `int`, then the compiler would warn about the comparison between `i` and `v.size()`
  - comparison between signed and unsigned int, which is tricky as discussed in Section 6.8.
  - By declaring `i` to be `size_t`, the warning is suppressed.

I want to make a technical remark about vectors. So there is a member function `size` which returns the size. Now, the size is returned as a type `size_t`. `Size_t` is just an alias for a certain kind of unsigned integer type. And it is a type which is meant to, it is a type or it is a type of variable, which is meant specially for storage array indices. Now, therefore, when you go through array elements or rather vector elements, you should use `size` type for the index variable. Because if you have a code like this, okay, here you are comparing the index with `v.size`. If you had declared this `i` to be `int`, then you would be comparing an integer with an unsigned integer. Now, this comparison is something that C++ finds a little bit error prone, because unsigned integers have that wider range and C++ compiler writers worry that look, “Are you sure you want to do this comparison”, that kind of thing. And therefore, instead of declaring this control variable to of `int`, if you declare it of `size_t` you will not get that warning about this comparison between an unsigned int and an int. So, yeah, so this is tricky as was discussed in 6.8 and therefore, the compiler puts a warning but you do not want, when you compile you do not want any warnings.

You do not want any clutter and therefore, it is a good idea to just use a `size_t` over here rather than an `int`. If you use `int`, it is not a big deal but it will just have clutter when you compile. Now, the vector class can also be used to build multi-dimensional vectors, but really I have to say multi-dimensional arrays if you will. Here is how you do it.

(Refer Slide Time: 11:18)



**Multidimensional vectors**

```
vector<vector<int> > vv;
```

- Each element of `vv` is itself a vector of `int`.
- We must supply two indices to get to `int`. Hence it is a 2d vector!
- Space needed between `> >`

```
vector<vector<int> > vv1(5, vector<int>(10,23));  
// vv1 has 5 elements each of which is a vector<int>  
// of length 10, having initial value 23.
```

- Note: syntax is not new, just specifying the length and initial value.
- `vv1.size()`, `vv1[0].size()` : number of rows and columns.

So you say `vector<vector<int>>`. Okay that is what this variable `vv` is. And note by the way that I need to put a space over here. Okay. If you put these `<<` consecutively then C++ will interpret as the output redirection operator. Okay, so what does this nominally say? It says, nominally says that each element of `vv` will itself be a vector of integer, right? Because after all what you put over here is supposed to tell you the type of the elements of this vector. So, each element here will be a vector of integer.

So, that means to get to the integer itself you must apply 2 integers, so `v[0]` will get me to a vector of integers. If I add another index I will get to the actual integer. So, we really have a 2 dimensional over here. So, here, for example, is a definition of something which looks complicated but which can be easily understood. So, I have a vectors of integers, that is called `v1`. So, this 5 tells me how many elements this vector has. Whatever follows is supposed to tell me the initial value of this element, okay but this element is itself is a vector of ints. So the initial value of vector if ints is going to be a vector of int of size 10 and each element having a value 23. So what have we done? Okay. So, `v1` has 5 elements, each of which is a vector of ints and each has length 10 having initial value 23. So what do we get at the end of this?

We are going to get a 5 by 10 array okay all elements 23. So, quite cool. Notice that this syntax is not really new. I already introduced this syntax, but I am just using it in a somewhat elaborate manner. Now, if I define a vector like this, I can get how many rows and columns in it, are in it by just asking the size of `vv1`. So, the size of `vv1` is 5, and if I go to the 0<sup>th</sup> element then it is going to be something like this and its size will be 10. So, I am using this to store a matrix, and if I want to get the size, the number of rows and the number of columns, I can just do something like this.

(Refer Slide Time: 14:26)

### Creating a 5x5 identity matrix

```
vector<vector<double> >  
    m(5, vector<double>(5,0));  
// m = 5x5 matrix of 0s.  
  
// elements of m can be accessed  
// specifying two indices.  
for(int i=0; i<5; i++)  
    m[i][i] = 1;  
// place 1s along the diagonal.
```

So by using vector of vectors, I can create a 5 by 5 identity matrix and here is how I am going to do it. So first of all, let us see what this does. Okay, so I am going to have a vector of vector double and I am going to call it m. It is going to have 5 elements, each of which is a vector of doubles of length 5, but all elements 0.

So at this point, already I will have a 5 by 5 matrix, but in this matrix every element will be 0. That is not quite what I want. So, but I can access the individual elements. So, what I do is, I am just going to go over the diagonal and for that I am going through i is equal to 0 to 4 and this will give the diagonal, so 00 11 22 33 all of these will be set to 1. So, now, I will have a 5 identity matrix.

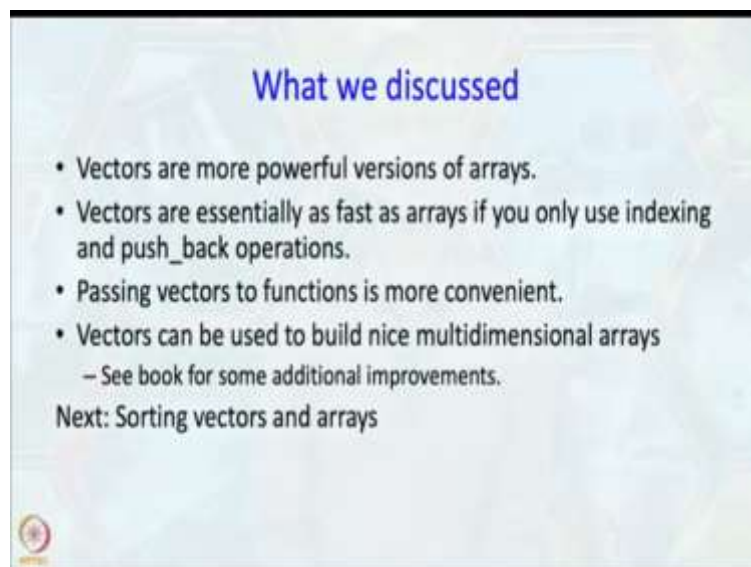
(Refer Slide Time: 15:18)

### Exercises

- Create a 4 element vector storing pointers to char.
- Write a function which multiplies two matrices stored as vector of vectors.
  - Should work for multiplying any two matrices, such that the number of columns in first = number of rows in second
  - Pass the arguments by reference
  - Return the result matrix

So here is an exercise, 2 simple exercises which I will definitely encourage you to do.

(Refer Slide Time: 15:25)



Alright, what have we discussed in this segment? So we have discussed – vectors which are more powerful version of arrays. Here is a point that you should note – vectors are essentially as fast as arrays, if you only use indexing and push\_back operations. If you insert in the middle then things get slowed down, but with just these 2 operations then practically arrays. Passing vectors to functions is more convenient because you do not have to specify the length.

You can get by doing v.size. Then vectors can be used to build nice multi-dimensional arrays. Okay. So in particular you should be able to write matrix multiplication operations or all kinds of matrix operations which are all matrix sizes. This is the end of this segment and in the next segment I am going to talk about sorting vectors and arrays but let me take a quick break.