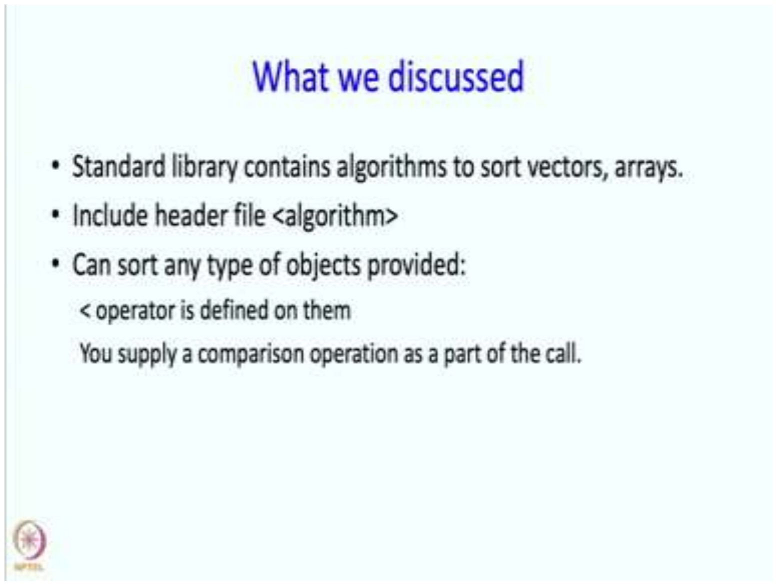
An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture-23 Part-03
The Standard Library
Classes map and unordered map

(Refer Slide Time: 0:18)

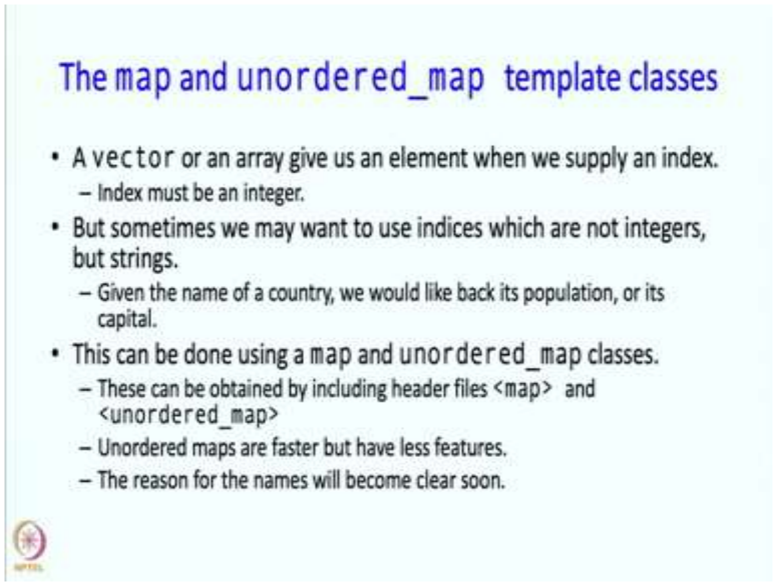


What we discussed

- Standard library contains algorithms to sort vectors, arrays.
- Include header file <algorithm>
- Can sort any type of objects provided:
  < operator is defined on them
  You supply a comparison operation as a part of the call.

Welcome back. In the previous segment, we discussed sorting.

(Refer Slide Time: 0:26)



The map and unordered_map template classes

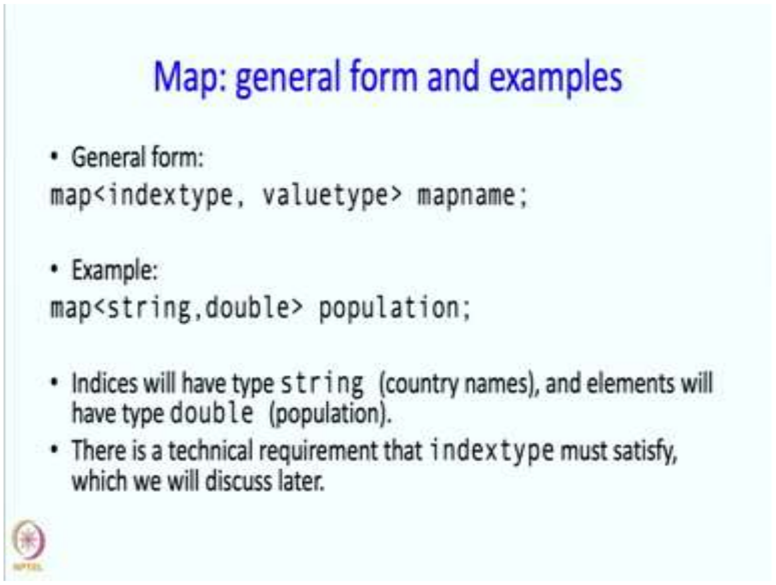
- A vector or an array give us an element when we supply an index.
  - Index must be an integer.
- But sometimes we may want to use indices which are not integers, but strings.
  - Given the name of a country, we would like back its population, or its capital.
- This can be done using a map and unordered_map classes.
  - These can be obtained by including header files <map> and <unordered_map>
  - Unordered maps are faster but have less features.
  - The reason for the names will become clear soon.

In this segment, I am going to talk about maps and unordered maps, so these are also template classes and will see details. So, let me describe them starting with vectors. So, a vector or an array, gives us an element when we supply an index. The index must be an integer and it must come from a small range. But sometimes we may want to use indices which are not integers, but maybe strings. So, for example: Given the name of the country, we might want to use the name of the country as an index and we would like back its population, or maybe we want back its capital.

Now, this is exactly what maps or unordered maps can do and these can be obtained by using header files map and unordered map. Unordered maps are faster but have some poor features. So we are going to talk about one feature that they have less, but they have some other features also. But if you do not need those features you should really use unordered map because they are faster and the reason for these names will become clear soon.

Well, the reason for the name map should be clear because the map is really in this example mapping the name of a country to the population or it is mapping the name of the country to a capital, so it is like a mathematical map. A mathematical maps elements of one set to elements of another set, so that is what this is doing. Unordered map will become clear a little bit later.

(Refer Slide Time: 2:19)



Map: general form and examples

- General form:

```
map<indextype, valuetype> mapname;
```

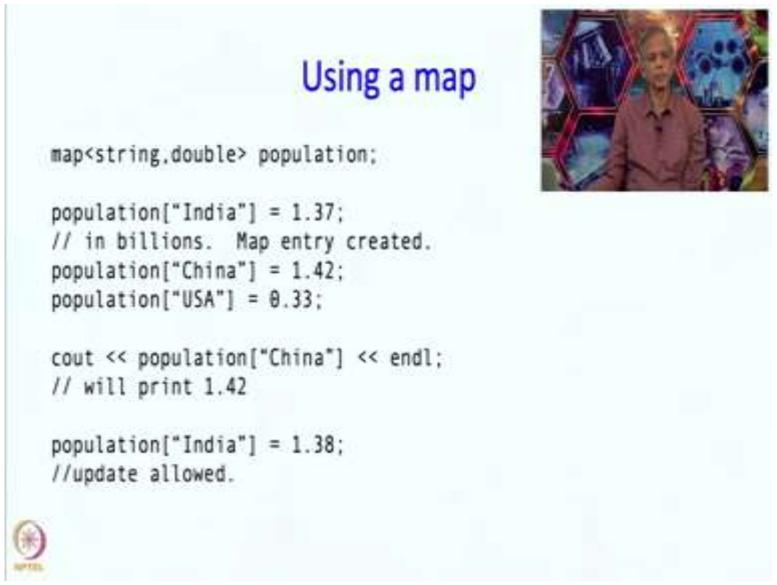
- Example:

```
map<string,double> population;
```

- Indices will have type `string` (country names), and elements will have type `double` (population).
- There is a technical requirement that `indextype` must satisfy, which we will discuss later.

So, let us talk about maps, so what is the general form and take some examples. So, if I want to define a map, here is what I write, I write the keyword map, then I write in braces the two template arguments, the index type and the value type and then I give the map name. So, as an example, I can write map<string,double> population. This is going to be useful for doing the thing that we just say given a country, let us get its population. So, here the indices will have type string, so, for example, I can have country names and elements will have type double so that could be population if I want. And if it is going to be a map there is a technical requirement that this index type must satisfy. So, well I might as well mention it later that the index type must be comparable using the less-than operator and how that actually comes in, we will discuss later. But how do we use this, how do we use a map.

(Refer Slide Time: 3:30)



Using a map

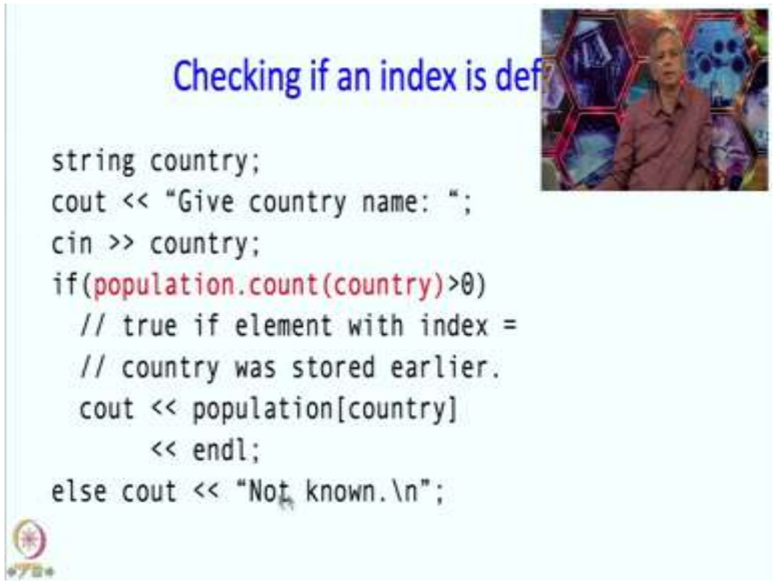
```
map<string,double> population;

population["India"] = 1.37;
// in billions.  Map entry created.
population["China"] = 1.42;
population["USA"] = 0.33;

cout << population["China"] << endl;
// will print 1.42

population["India"] = 1.38;
//update allowed.
```

So we have a map from strings to doubles and we our map is called population. So, here is what I can do, I can say population index India is 1.37. So, this actually creates an entry in the map, it looks like an array access statement. But it is a lot more interesting, it is creating an entry and then the index does not have to be an integer in a small range. But it can be anything, well it can be string as we have specified over here and it is storing 1.37 in that element of the population.

So, map entry or map element has been created now. I can create more and more, and I can print out what is stored in the map. So, if I write cout<<population["China"], this will print out 1.42. I can update the entries as well. So suppose, I just realized that this was an old entry and the new population is slightly larger, then I can put this new population. 1.37 will go away and it will be

replaced by 1.38.

(Refer Slide Time: 4:58)



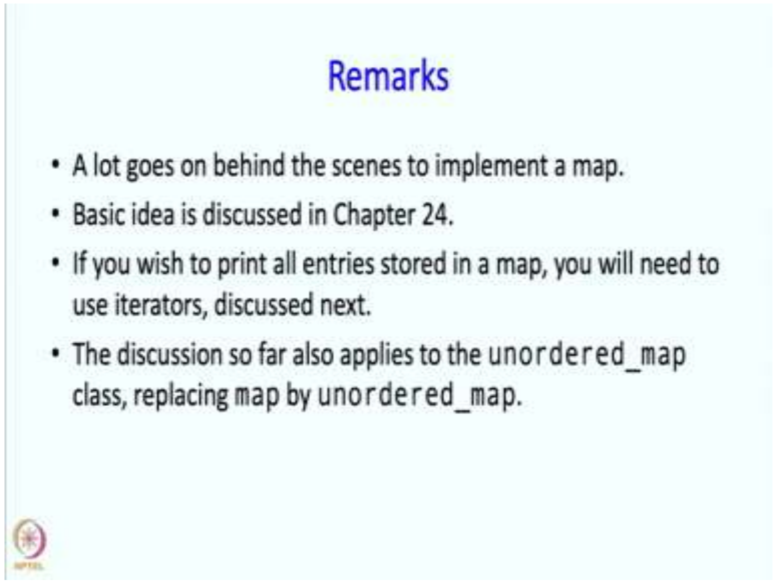
Checking if an index is def

```
string country;
cout << "Give country name: ";
cin >> country;
if(population.count(country)>0)
  // true if element with index =
  // country was stored earlier.
  cout << population[country]
       << endl;
else cout << "Not known.\n";
```

Now, you would like to know whether our map contains say a certain country. So, I would like to check if a certain index is defined. So, say we are doing lookup, a population lookup service, so somebody types the country and now, I want to know whether I have the population with me. So for that, I am going to write population.count, so count is a member function on population on those specific maps it takes a country as an argument and it says look is there an element with this country as the index whatever you type done.

So, for example, if we had placed the population of India, China and US and suppose you typed in Sri Lanka then this population count what turn out to be 0. Otherwise, it will be 1. And by the way, there is a multi-map as well, which allows you to put several entries and that is why this can be a general number, but with maps, it has to be either 0 or 1. So, if this entry has been defined then we can say print out the population and if this entry is not defined, we can say the population is not known. So, this is a very short population lookup service that you can put up.

(Refer Slide Time: 6:13)

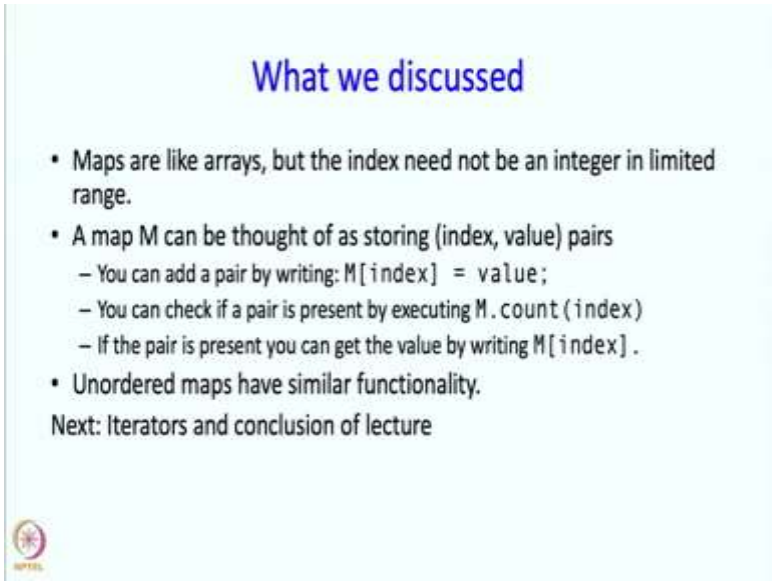


Remarks

- A lot goes on behind the scenes to implement a map.
- Basic idea is discussed in Chapter 24.
- If you wish to print all entries stored in a map, you will need to use iterators, discussed next.
- The discussion so far also applies to the `unordered_map` class, replacing `map` by `unordered_map`.

So, as you might expect a lot is going on behind the scenes and there is actually a very interesting idea and that idea is discussed in chapter 24. It is a bit of an advanced idea, so we are not going to consider it in this chapter. If you wish to say, that look I have stored a lot of data in the map and I want to print it out. I want to print out all the countries and their population that I know. So you can do that and that will need to use of iterators which we are going to discuss next. The discussion so far also applies to unordered map, so everything that I had said over here just replace map by unordered map and it will work.

(Refer Slide Time: 7:02)



What we discussed

- Maps are like arrays, but the index need not be an integer in limited range.
- A map M can be thought of as storing (index, value) pairs
  - You can add a pair by writing: `M[index] = value;`
  - You can check if a pair is present by executing `M.count(index)`
  - If the pair is present you can get the value by writing `M[index]`.
- Unordered maps have similar functionality.

Next: Iterators and conclusion of lecture

Alright, so what have we discussed? We said that maps are like arrays, but the index need not be an integer in a limited range. A map M and this applies to an unordered map as well can be thought of as a pair, so you are storing these pairs, the index and the value pairs. And the way to add a pair is this, so you say M[index]=value. This adds, this pair into the map.

And you can check if a pair is present by writing M.count we saw this and whether checking with it is greater than 0 or not and you can get the value by writing M[index]. Unordered maps have similar functionality. And this is the end of this segment. In the next segment, I am going to talk about iterators and we will conclude this entire lecture sequence. Thank You!