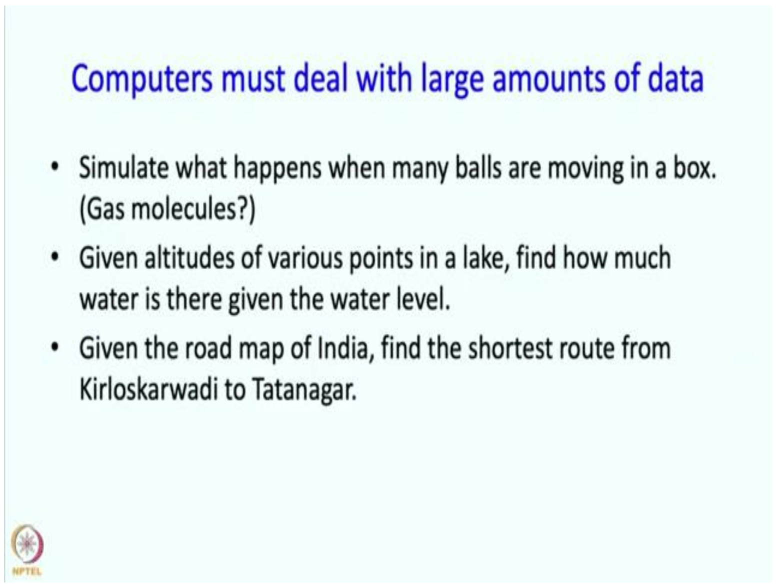
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Lecture No. 15 Part - 1
Array Part – 1
Introduction

Hello and welcome to the NPTEL course on An Introduction to Programming through C++, I am Abhiram Ranade and this lecture is going to be on arrays. The reading for this is chapter 14 of the text.

(Refer Slide Time: 0:48)



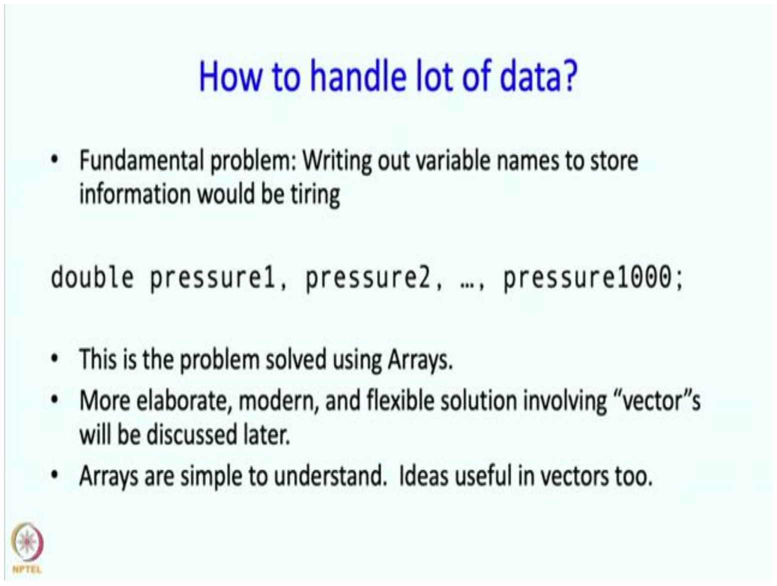
Computers must deal with large amounts of data

- Simulate what happens when many balls are moving in a box. (Gas molecules?)
- Given altitudes of various points in a lake, find how much water is there given the water level.
- Given the road map of India, find the shortest route from Kirloskarwadi to Tatanagar.

So let me begin with an obvious observation, so computers should generally deal with large problems, large computations and large data. So for example, we may want to simulate what happens when many balls are moving around in a box? Of course, this is not, these are not real balls, they might be gas molecules and we maybe simulating a gas and what kinds of effects it has on the balls or pressures and things like that.

Given altitudes of various points in a lake, find how much water is there given the water level. So this might well be a computation that we might want to do. Or given the road map of India, find the shortest route between two cities and this is probably something that you might already have used, you might already have used a trip planning program.

(Refer Slide Time: 1:36)



How to handle lot of data?

- Fundamental problem: Writing out variable names to store information would be tiring

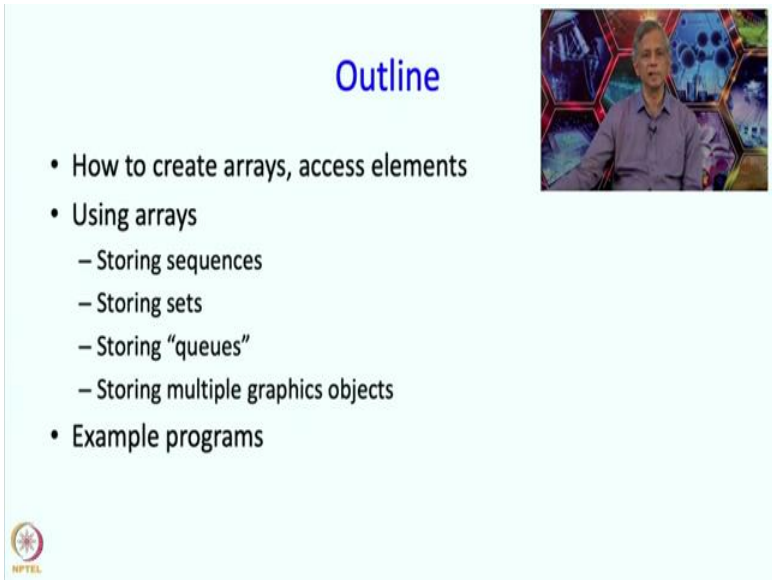
```
double pressure1, pressure2, …, pressure1000;
```

- This is the problem solved using Arrays.
- More elaborate, modern, and flexible solution involving "vector"s will be discussed later.
- Arrays are simple to understand. Ideas useful in vectors too.

So computers handle a lot of data and how do they do that? So there is a very very basic problem. If you want to have lots of data you want to have lots of variables presumably. So, if you need lots of variables, then just writing out the variable names to use would be really tiring. So maybe you want 1000 variables to represent pressures at 1000 points, then you may have to write something like this. "double pressure1, pressure2, pressure1000", nobody would really want to do that.

So arrays solve this problem and actually a little bit more. And arrays come to us from C which is the precursor of C++ and in C++ there is a more modern, a somewhat nicer solution to this problem called a vector class, so we will discuss this a little bit later. But, arrays are in some sense simpler to understand and they do most of the work anyway so whatever we are going to do in this lecture sequence will be definitely useful in any case.

(Refer Slide Time: 2:58)

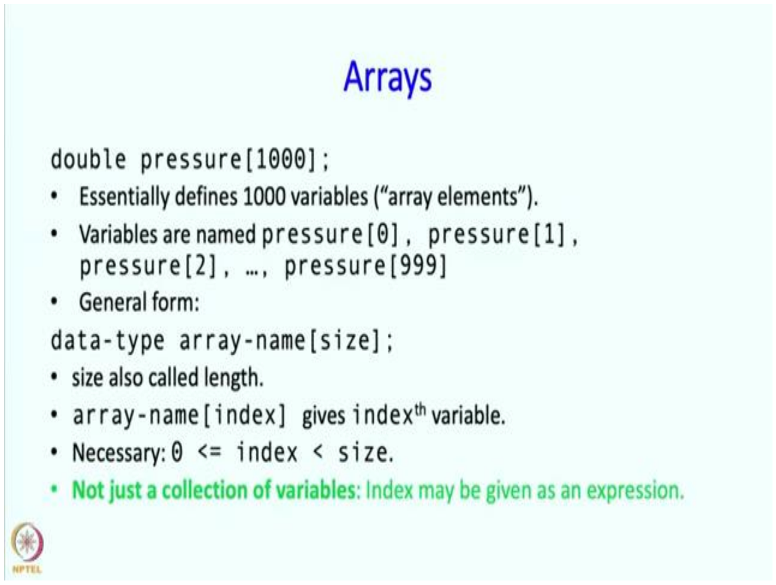


Outline

- How to create arrays, access elements
- Using arrays
  - Storing sequences
  - Storing sets
  - Storing "queues"
  - Storing multiple graphics objects
- Example programs

So here is what we are going to do, we are going to talk about how to create arrays and how to access their elements. And then why do we use arrays? So well we are going to use arrays for storing sequences, storing sets, storing something which we might call a queue and storing multiple graphics objects. And we will do lots of example programs along the way, so that is, that is the outline for this lecture sequence.

(Refer Slide Time: 3:28)



Arrays

`double pressure[1000];`

- Essentially defines 1000 variables ("array elements").
- Variables are named `pressure[0]`, `pressure[1]`, `pressure[2]`, …, `pressure[999]`
- General form:

`data-type array-name[size];`

- size also called length.
- `array-name[index]` gives indexth variable.
- Necessary: `0 <= index < size`.
- **Not just a collection of variables:** Index may be given as an expression.

So, let me begin with a definition of arrays or rather an example. So, if I want to define the pressure variables which I mentioned in the last slides, in the last few slides, then I just need to write this. Now this essentially defines 1000 variables and these 1000 variables are called array elements and together those variables comprise what is called an array. And the variables you can think of as we named not pressure0 onward but pressure[0] as the first variable.

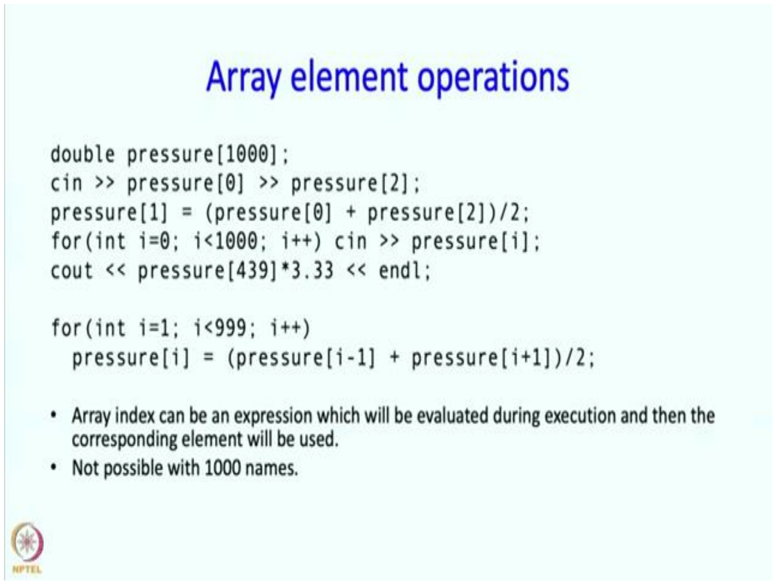
pressure[1] as the second variable and so on, till pressure[999]. That is effectively the name of the last variable we are creating, using this statement. Notice that the numbering starts at 0 and not at 1, so the variables are not named pressure 1 through pressure 1000, but they are named pressure 0 through pressure 999, all within the indices, the 0, 1 to 999 are in square brackets.

So the general form is the type of the element data type, array name and the size, the size is also called the length. So in our previous example the data type was double, every element was of type double, the name of the entire array was pressure and it had a 1000 elements. And the moment you say it has 1000 elements you are also saying that the numbering goes from 0 to 999.

And it is customary to say that array name[index] gives the indexth variable, where of course index goes from 0 to 999 or in general index goes from 0 to size-1, index has to be smaller than size. Now, arrays actually end up being a little bit more than just a collection of variables.

So this is because the index can be given as an expression, so I do not have to say 999 or I do not have to say 47, I can put another variable name there and the value of that name will be taken during execution, so this makes arrays really powerful and not only variable names, but I can put in whole expressions there as I wish. And we will see examples of all of these things.

(Refer Slide Time: 6:28)



Array element operations

```
double pressure[1000];
cin >> pressure[0] >> pressure[2];
pressure[1] = (pressure[0] + pressure[2])/2;
for(int i=0; i<1000; i++) cin >> pressure[i];
cout << pressure[439]*3.33 << endl;

for(int i=1; i<999; i++)
  pressure[i] = (pressure[i-1] + pressure[i+1])/2;
```

- Array index can be an expression which will be evaluated during execution and then the corresponding element will be used.
- Not possible with 1000 names.

So, what can you do with arrays? Well suppose we have declared our array in this manner, then I can read into its elements. So, if I say cin and instead of putting the name of some simple double variable I put in this pressure[0], which is also a variable., so this would cause whatever the user types, the first thing to go into pressure[0] and the second thing to go into pressure[2].

So you can use array elements basically wherever you can use ordinary variables. So I could write this for example, this is just saying that pressure[1] so the element of the array pressure at index 1 is assigned value which is the mean of the values of pressure[0] and pressure[2]. And as I said we could have the index be a variable, so here I have a for loop with the control variable i going from 0 through 999 and we are reading consecutive words that the user types into pressure[0], pressure[1], pressure all the way till 999.

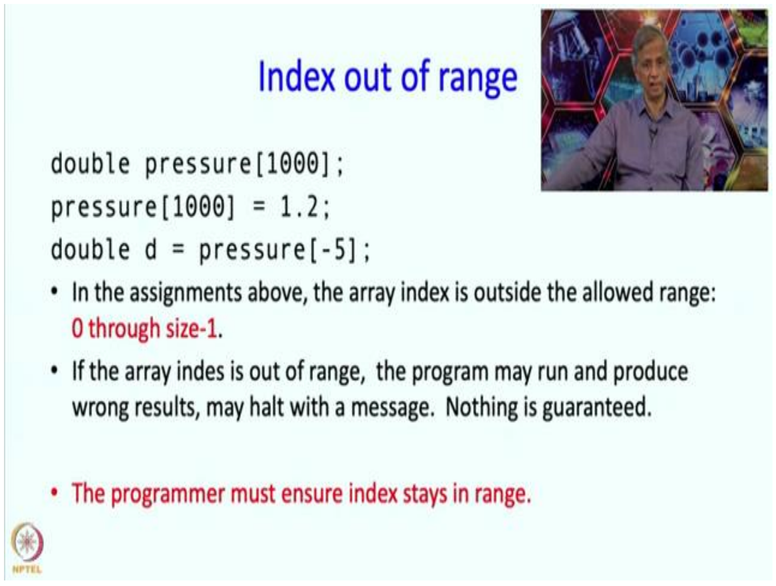
So notice that we, this could not be written in general if we were, if we had to write a constant in place of that square bracket, i square bracket, so it is really important that that number be coming out of a variable. So in any case this will take the 1000 values that the user types and place those 1000 values in pressure 0 through pressure 999, in the 0th iteration the value received will be placed in pressure[0], then the value of i will increment, the next value received will be placed in pressure[1] and so on.

And I can print out the values as well and again I could have written something like p*3.33 and for any variable instead of any variable I can put in any array element reference. Here is another interesting statement, now this time i is going from 1 and not to 999, to 998, so what is this doing? So it is saying pressure[i]=(pressure[i-1]+pressure[i+1])/2.

So the pressure in every element is made equal, is being made equal to the average of the pressure in the previous and the next element, this is not, this is not a real computation, but I am just giving this as an example to show what kind of expressions we can write. And here I cannot use the array index i to start at 0, because if I did that then for i equals 0, this value would be 0, but this value would be minus 1 and that is not allowed. So we want the index to always be between 0 and the size, so less than the size. So the size is 1000, so this had better be less than the size and so had this be. So, if in particular i had been 999, then this would be, this would be 1000 and that would not be allowed. So therefore, we have carefully chosen the bounds of this loop to be 1 and 999.

So the important point which I have noted is that the array index can be an expression, say for example, here i-1 and i+1 or even just simple plane i is an expression, which will be evaluated during execution and then the corresponding element will get used. Now, if we just had written pressure[0], pressure[1], pressure[2]without the square brackets, we actually had taken the pain of writing down all the 1000 names, we would not be able to write something like this loop. Because over there, if your variable is pressure[0] one word then you cannot, you cannot replace a part of a name by the value of a variable. So this is this, this square bracket notation and the way we define arrays is really important.

(Refer Slide Time: 11:04)



Index out of range

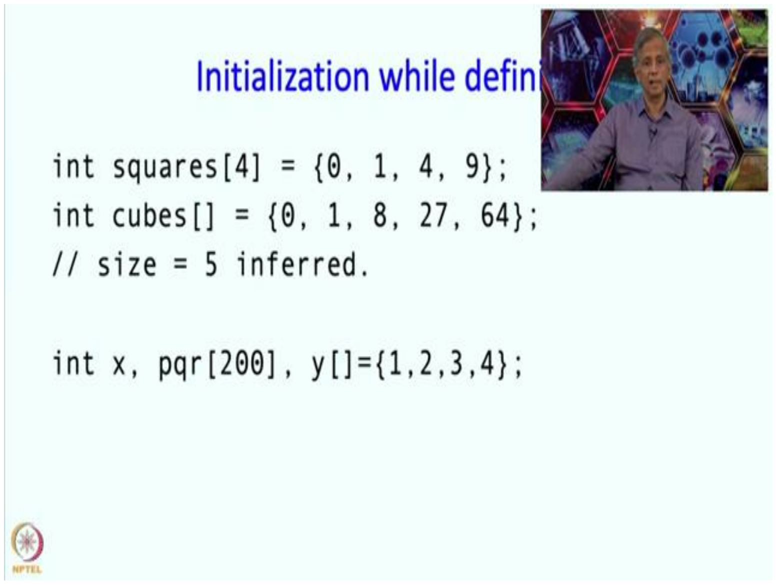
```
double pressure[1000];
pressure[1000] = 1.2;
double d = pressure[-5];
```

- In the assignments above, the array index is outside the allowed range: 0 through size-1.
- If the array indes is out of range, the program may run and produce wrong results, may halt with a message. Nothing is guaranteed.
- The programmer must ensure index stays in range.

Now here we have again the same definition and suppose I write pressure[1000]=1.2, to look at it, it does not look bad. However, we said that the index or whatever appears in these square brackets must be in range and the range in 0 through 1 less than the number of elements. So this number is supposed to be, supposed to be smaller than 1000. So, if you put a 1000 over here that is not good, that is a mistake, that is an error.

So here I have put the number minus 5 and that is also a mistake, so both of these mistakes are said to be caused because the array index is outside the allowed range. And if the array index is out of range, then it is a mistake and nothing really is guaranteed, the program may run, but of course it cannot run correctly and so it may produce wrong results or it may halt with a message. So you really have to be very careful when using arrays, you have to, you have to be very sure that the indices are within the specified range.

(Refer Slide Time: 12:32)



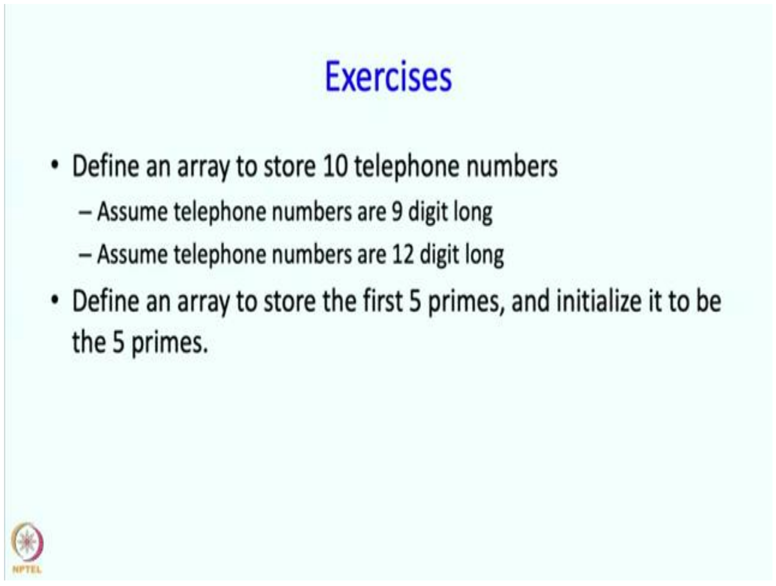
Initialization while defin

```
int squares[4] = {0, 1, 4, 9};
int cubes[] = {0, 1, 8, 27, 64};
// size = 5 inferred.

int x, pqr[200], y[]={1,2,3,4};
```

Now, when you define arrays you can initialize them if you wish. So for example, this is an integer array called squares with 4 elements and I want those elements to be 0, 1, 4 and 9. So I can write them in this manner, so squares[0] will be 0, squares[1] will be 1, squares[2] will be 4, squares[3] will be 9. Or, I have put down cubes and I have not specified the size, but C++ will look at how many values we have specified and make cubes have that size. So here the size 5 will be inferred by C++ even if you have not put it down and you can mix things. So in the single statement I am defining an ordinary variable x, then I am defining an array pqr with 200 elements and I am defining an array y with 4 elements, but I have not specified its size and C++ will infer it.

(Refer Slide Time: 13:41)

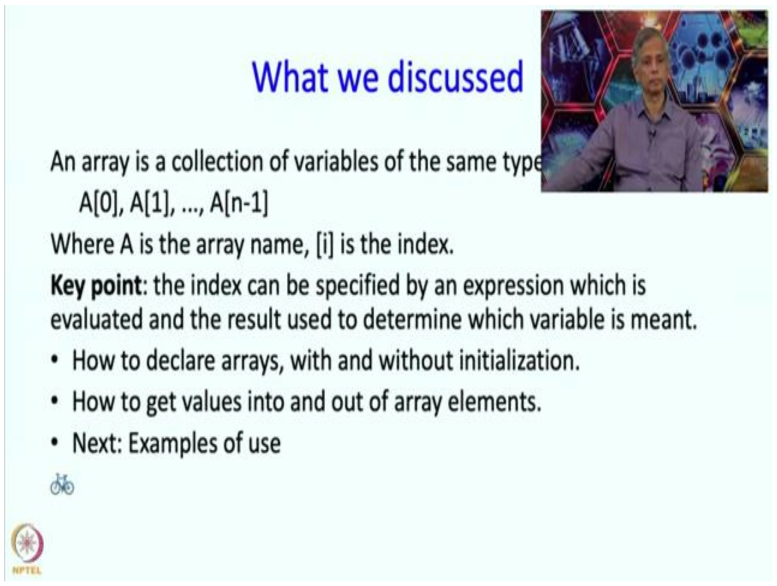


Exercises

- Define an array to store 10 telephone numbers
  - Assume telephone numbers are 9 digit long
  - Assume telephone numbers are 12 digit long
- Define an array to store the first 5 primes, and initialize it to be the 5 primes.

So some quick exercises for you, so define arrays of in various ways, so please do try them.

(Refer Slide Time: 13:51)



What we discussed

An array is a collection of variables of the same type

A[0], A[1], …, A[n-1]

Where A is the array name, [i] is the index.

Key point: the index can be specified by an expression which is evaluated and the result used to determine which variable is meant.

- How to declare arrays, with and without initialization.
- How to get values into and out of array elements.
- Next: Examples of use

So, what have we discussed so far? We have said that an array is a collection of variables of the same type. So if the number of elements in the array is n, then the array, the array, the elements are named A[0], A[1], A[n-1]. So A is the array name, and i is the, i is called the index. Now the important point is that the index can be specified by an expression, and the expression is evaluated and the result is used to determine which variable of these of this collection you mean.

So we saw how to declare arrays, with and without initialization. How to get values into and out of array elements. And next we are going to take some examples of use of arrays. So we will take a quick break.