Introduction to Problem Solving, and Programming
Prof. Deepak Gupta

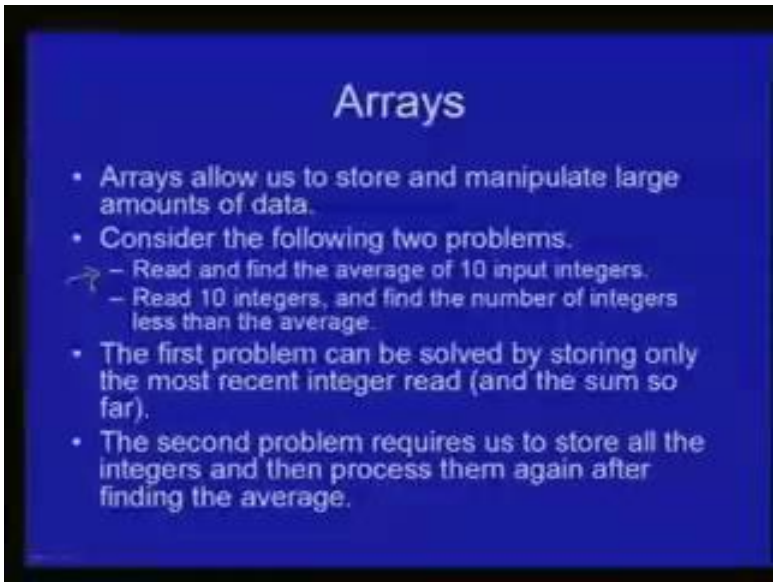
Department of Computer Science Engineering

Indian Institute of Technology, Kanpur

Lecture – 12

In today's lecture, we start talking about a very important concept in programming that is a notion of arrays arrays are very useful for storing, and manipulating large volumes of data. So, as an example to motivate the use of arrays. Let us consider the following two problems the first problem is very simple you are required to read 10 integers, and find their average.

(Refer Slide Time: 00:40)



.

We know how we how we can do? That we can write a simple loop, and in each iteration of a loop, we read an integer, and add it to the sum, and at the end of loop we just simply divide the sum by number of integers, and that gives us the average. The important point is that in this algorithm, we do not need to store all the integers at that we have read at the same time what we need to remember or store is only the most recently read integer.

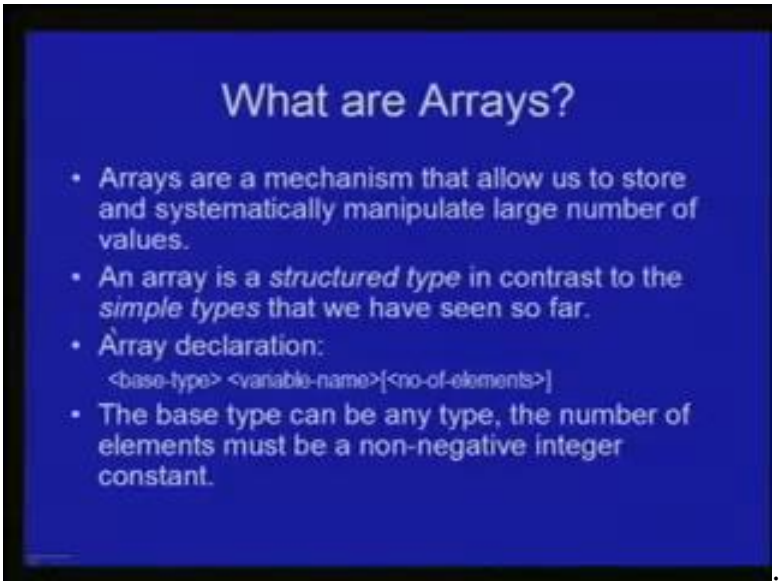
Now, contrast that with the second problem where we are supposed to again read the see read the integer find the average, but now, find out how many integer in the input were less than the average. Now, if you have to solve this problem will have to first find the average as before, but, then we will have to look at the each number again, and compare

it with the average to find out whether it is less than or equal to or more than the average, and count the number of integers which are less than the average.

This means that we cannot forget an integer soon after reading it we must remember this integer we must store its value in some variable. So, that later it can be retrieved. Now, without the use of arrays we could solve this problem by having may be 10 integer variable, but, then the problem will be that will have difficultly writing that loop, because the variable name will change for every iteration of the loop.

The arrays allows us to solve this problem very elegantly as we will see soon. So, as we seen an array is essential mechanism that allows us to store, and systematically manipulate large number of values.

(Refer Slide Time: 02:19)

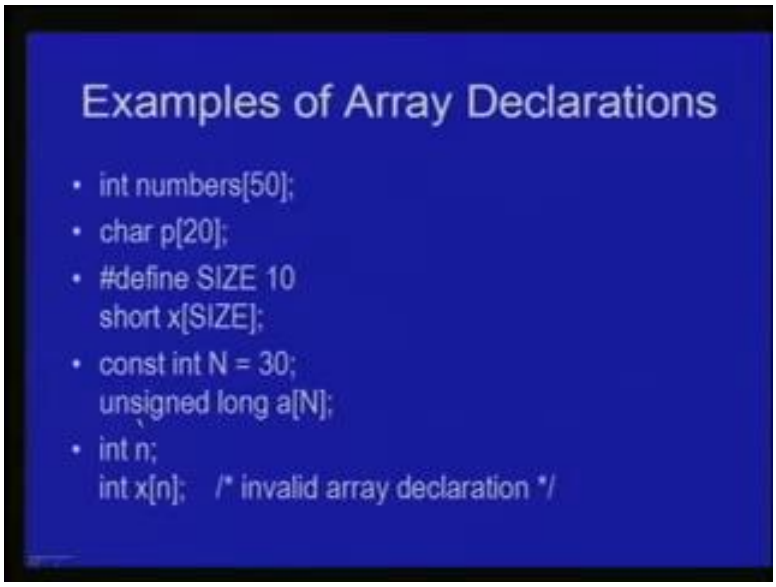


An array is what is known as structured type in contrast. So, as i mention there is a mechanism that allow us to store, and systematically manipulate large number of values. So, in contrast to the simple types that we have seen. So, far an array is a structured type that is it an array is composed of values of simpler types, and this is how we declare an array. We'll soon see number of examples here is the first what we have to write is the base type the base type is the type of the individual elements of the array a array is com an array is composed of a large number of individual elements which are of some type they all have to be of the same type.

So, this base type is the type of the element of the elements of the array, and then the next variable name is the actual name of the variable that represents the array, and then within the square brackets we have to mention how many elements of this base type will the array have, so the base type can be any type that we have seen. So, far the number of elements must be in non negative integer constant that is the compiler at compile time must be able to figure out how much how many element are there in the array, and therefore, how much space in the memory is to be allocated for that array.

So, let us see some examples of array declaration. So, here is the simplest example into here is the base type the name of the array or the variable name is number, and 50 is the size of the array, and this declaration essentially says that numbers is an array which will be comprising of 50 integers.

(Refer Slide Time: 05:08)



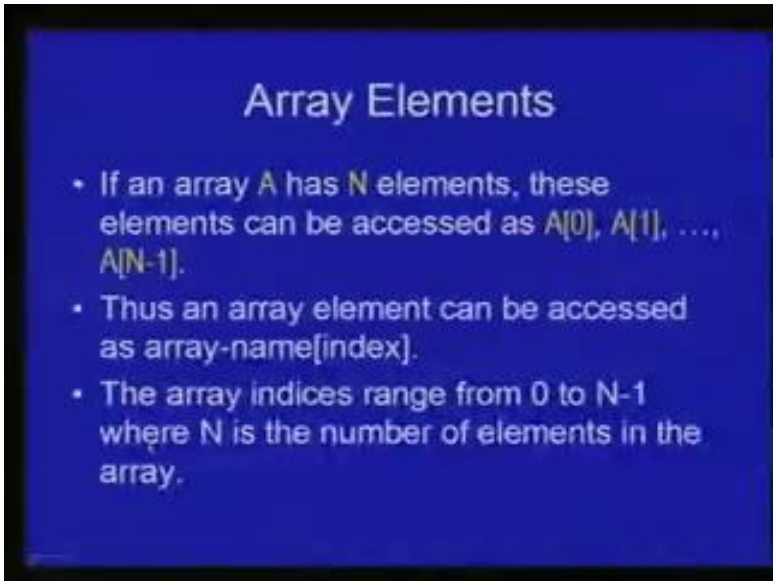
Here is another array. So, this time the array name is p number of elements is twenty, and the type of each element is char this shows that the base type can really be any type. This in this next example the base type is short, and instead of giving the size as a literal constant here used a symbolic constant which we define using this hash define which would be familiar to you already from earlier examples.

So, the size size of the array need not be a literal constant, but it has to be a constant value whose value can be determined by the compiler at compile time. Here is the next example in this case we have declared the size of the array to be kept as N, where N is

declared to be a constant in different way the style of declaration constant is also that something we have already seen, and the base type in this example is unsigned long.

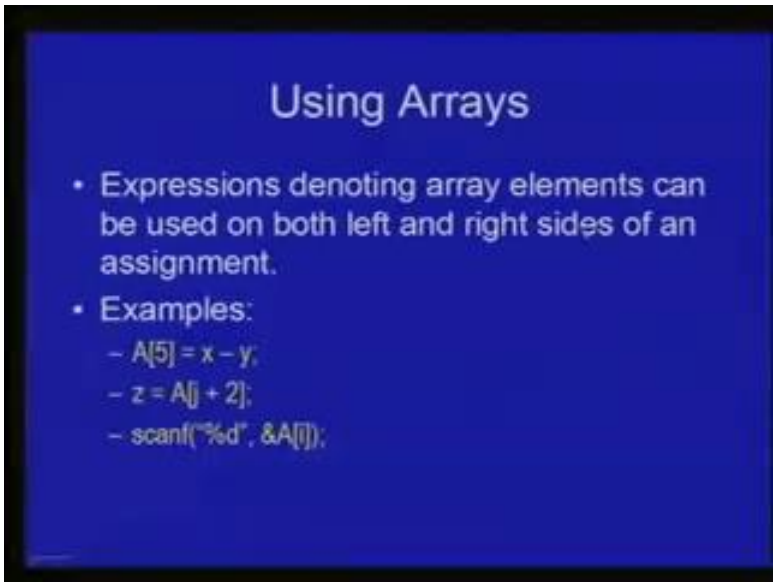
Here is an example, in which the array declared declaration is invalid that is if we have such an array declaration in the program, and the compiler will give an error, and the reason is that the size of the array has been declared to be n which is the name of the variable here. So, this is not allowed, because as i said the size of the array has to be a constant which is known at the compile time. Let's now, talk about how we can access, and manipulate the individual array elements. So, if an array we have declared to have N elements, then these array elements these n array elements can be accessed as a 0 a 1 a 2 up to a n minus 1.

(Refer Slide Time: 05:29)



So, essentially the array element is accessed by giving the name of array a for example, and then within the square brackets the number or the index of the array element that we wish to access, and if an array has n elements, then the index will range from 0 to n minus 1. So, we can have expressions like this which denote array elements, and such expressions can be used both on the left, and the right side of an assignment.

(Refer Slide Time: 06:13)

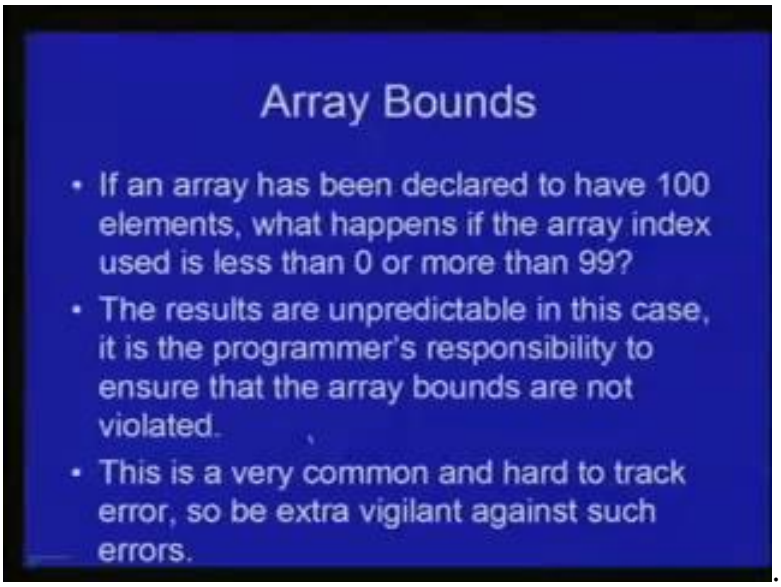


Here are some examples, so this assignment statement is assigning the value x minus y to the fifth array element remember that the array element start from the 0 element, and go up to the n minus 1 element where n is the size of the array. This in the second example the array element A j plus two occurs on the right side of the assignment, and the other thing you should notice about this particular assignment is that the array index array element index which is being used here is not a constant it is j plus 2 where j could be the j could be any variable whose value will known only at runtime.

So, only while declaring the array the size has to be declared as a constant, but while accessing individual elements of the array the indices used frame the arbitrary expression which result in an integer. Here is one more example showing how an array element can be read from the input we are using familiar scanf library function for doing that, and the the way we are doing that is very similar to the way we read a normal integer variable.

When we read an integer variable, we say scanf percent d ampersand x if we want to read in the value of highest array element, then all we have to say is instead of ampersand a x ampersand a i. Now, it's very important in programs to be clear full about the indices that we are using for the array elements as I have mentioned earlier if the array has been declared to have n elements, then the array indices range from 0 to n minus 1.

(Refer Slide Time: 07:50)



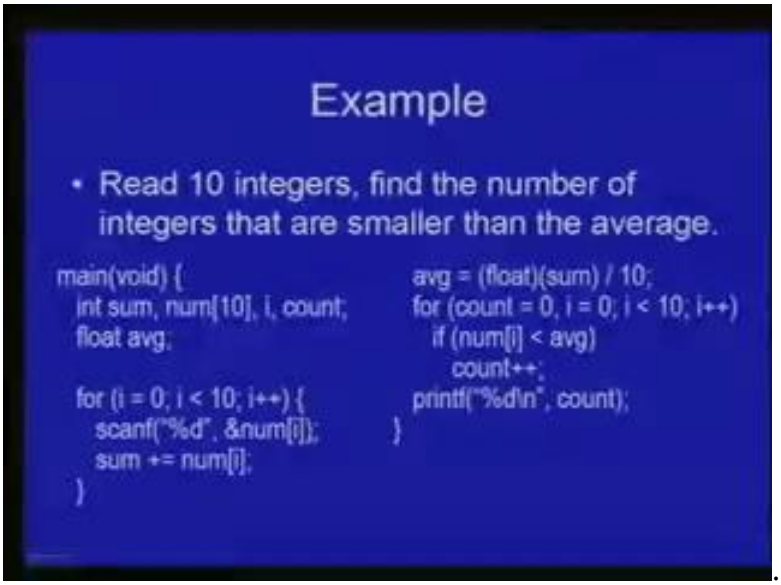
It is very important to ensure that the array element that we are accessing has an index between 0, and n minus one. So, for an example suppose we have declared an array to have 100 what will happen if the array index used in a particular expression is less than 0 or more than 99. The answer is that in this case the result of the program execution will be un predictable we will not be able to say what will really happen this is an error, because we are not allowed to use array indices less, than 0 or more than 99 in this case where the array has hundred elements, but the C compiler does not check for these kinds of errors.

So, what will happen at run time is some predictable, and un accepted behavior therefore, it is a programmers responsibility to ensure that the array bounds are not violated, and violation of array bounds actually is a very common problem in many programs many programmers make the mistake of accessing array elements which do not exist, and such errors do not necessarily result in the program crash they can have other un expected effects.

So, therefore, and these errors are very hard to track down therefore, it's we need to be very clear full while dealing whether there is that the expression that we are using for an array index is within the bounds for an array. So, now, here is first simple example program using an array, and this is the problem that i mentioned earlier while motivating

the use of arrays we have to read 10 integers, and find the number of integers that are smaller than the average.

(Refer Slide Time: 09:34)



So, we start the declaration of variables the variable sum is the one we are used to store the sum of all the integers that we read num 10 is the array which we will use to store all the integers that we have read. We call that once we have computed the average by summing up all the integers, and dividing the number. So, integers will lead to once again look at the each integer that we read, and compare it with the average to find out whether it is less than the average or not.

The variable count will be used to count the number of elements which finally, turn out to be less than the average, and the variable avg ought standing for average will contain the average, and this time in general not be an integer and. So, therefore, we have used the float type to store the average. So, this loop is as before. So, you are trying to read 10 integers, and find their sum. So, this for loop runs 10 times for values of i from 0 to 9, and we each iteration of the loop we read the highest element of the array, and also add it to the sum.

After that once we have found out the sum the average is nothing, but the sum divided by ten, but we have to be a little careful here, because remember that the sum the variable sum is of type into, and the literal constant n is also of type into which means that if we just divide sum by 10 we will get we will do an integer division which will mean that the

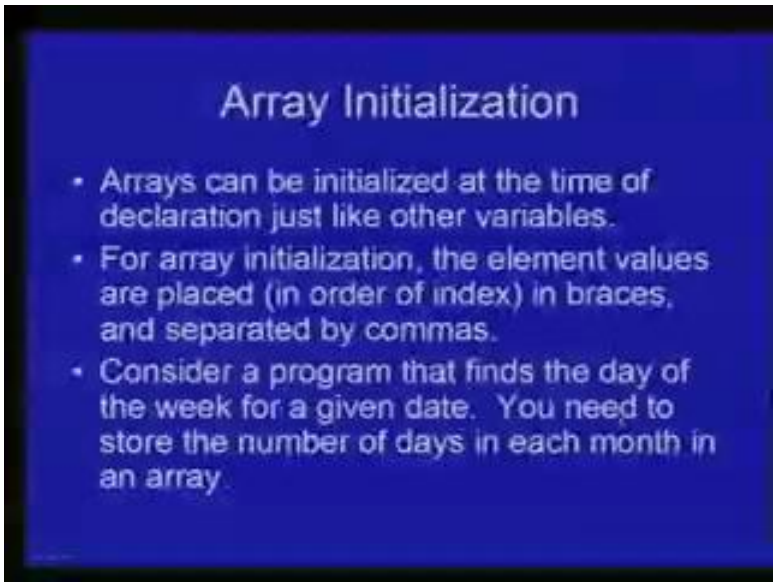
fractional part of the result will be dropped, and the final value that we will always get will be an integer.

So, therefore, to avoid that we are first typecasting sum into a float. So, that one of the operands of the division is now, a float and. So, automatically the other operand which is 10 will also be typecast to a float by the compiler before the division, and the answer will retain both the integer, and the fractional part of the result of the division.

And finally, having found out the average this loop is being used to find out the number of elements in the array which are less than the average note that we have again used the comma operator from the previous lecture to make two initialization in the for loop. We are initializing count to 0 remember count is the number of element that we that are found to be less than average, and i is also assigned to 0 this is the loop variable this will run from 0 to 9 when i becomes equal to 10 the loop will terminate.

So, within each iteration of loop if the highest element that we read or the highest integer we read is less than the average we add one to the count, and note again that this entire statement is a single if statement, and therefore, we do not need braces around it to make it the body of the for loop, and finally, this statement is outside the for loop this will be executed once the for loop finishes, and this simply obtains the value of the variable count which is nothing but the number of integers that have been found to be less than the average. Next come to array initialization as you know when we declare variables we can initialize them with the value at the same time the same thing can be done with arrays as well, and array initialization is quite simple.

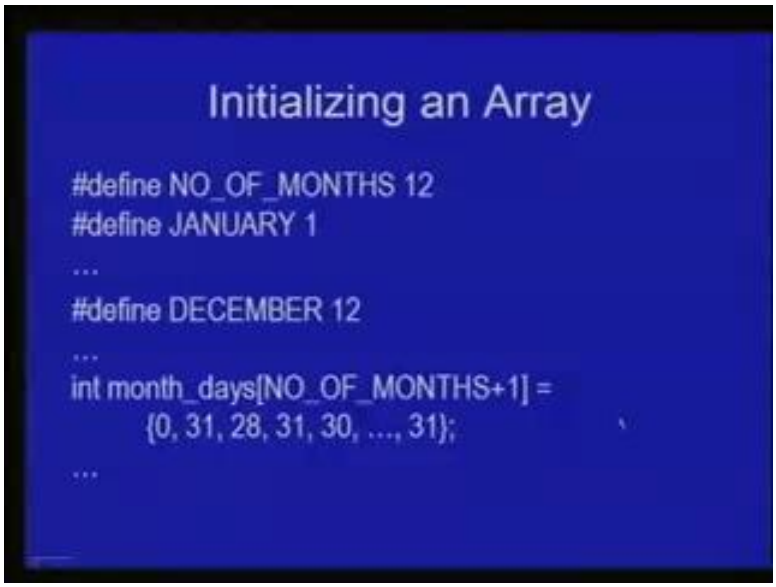
(Refer Slide Time: 13:31)



.

All you have to do is that the element values of the array in the order of their indices have to be placed in braces, and they have to be separated by commas. So, we will see an example of the use of such kind of an array initialization suppose we are writing a program that finds the day of the week for a given date. So, what we need to do probably is to store the number of days in each month in an array, and since these are constant values we can initialize this array right at the time of declaration. So, our program might something look like this. So, we define a symbolic constant for the various month from January to December January is the one, and December is defined as 12.

(Refer Slide Time: 14:04)

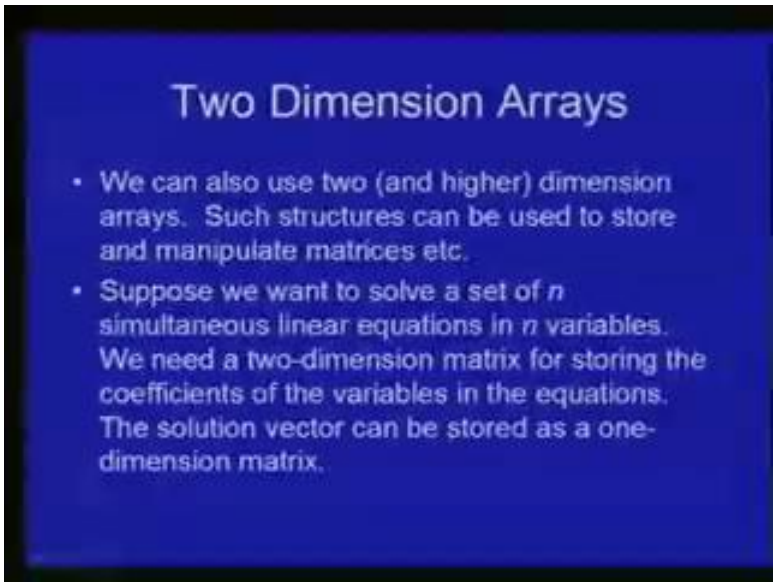


The number of months is of course, 12, and then we have this array month days which stores the number of days for each month now, we want to refer to January month January with an index one in this array let us say we could of course, have also used 0 for January, and eleven for December, and so on, but that might have become a little unintuitive so, but recall that when we declare an array to be of size n, and its indices range from 0 to n minus 1, and not from 1 to n, and since we need indices from 1 to 12 in array we need to make the array size one larger than the number of months, and So, therefore, the arrays size is number of months plus one which is of course 13.

So that means, that the array has elements at indices 0 to 12 the index the element at the index 0 will not be used. So, we are assigning it an value 0, and the rest are the number of days in the various months from January to December ignoring of course, the leap years. So, as you can see these array values initial values of the array elements have been placed in the order of array indices, and they have been separated by comma, and they have been enclosed in these curly braces.

So, this is how you can initialize an array next we come to higher level arrays or two dimension or higher dimension arrays; these are very useful for manipulating structures like matrices, and so on and so forth.

(Refer Slide Time: 15:58)

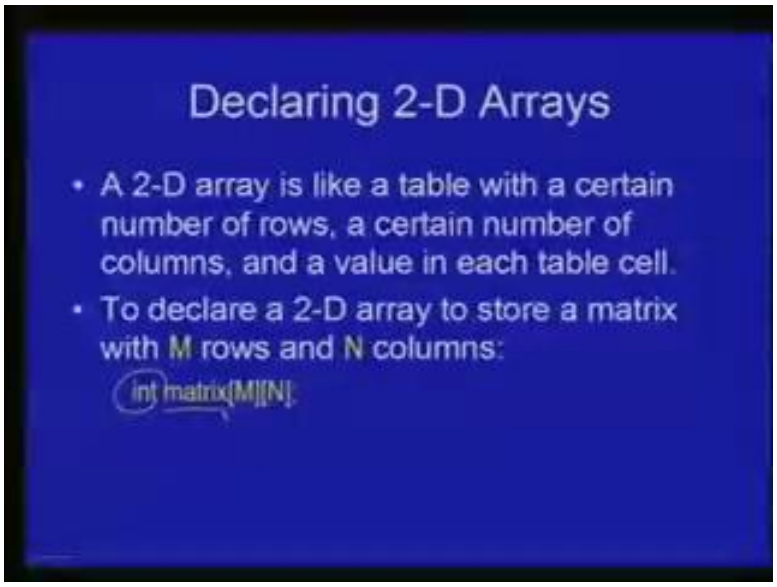


So, as an example, of this very common emerging problem suppose you want to solve a set of n simultaneously linear equations in n variable, then we need to store the

coefficient occurring in the n um equation. So, these can be easily stored in mathematics we represent them as an m cross n matrix. So, in c they can be easily stored as a two dimension matrix or a two dimensional array, and the solution vector can be stored as a one dimension array as usual.

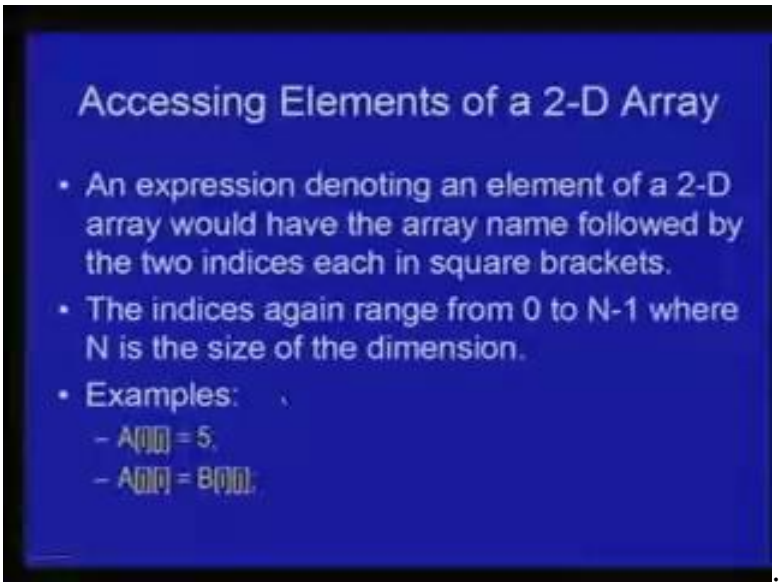
So, let us see how we can declare, and use two dimensional array. So, conceptually a two dimensional array is just like a matrix or a table with a certain number of rows, and certain number of columns, and one value at each cell of the table of the matrix, so, when we declare an array of a two dimension or indeed for higher dimension again what we need to is to specify the base type of the array element the name of the array in this case matrix the base type in the example is a into is into, and then we have to give the number of rows, and the number of columns both in square brackets one after the other.

(Refer Slide Time: 16:58)



The first is the number of the rows, and second one is the number of columns, where again the number of rows, and number of columns again have to be constant whose values can be determined at the compile time. So, in this example, the number of the rows in this matrix is m, and the number of columns in this matrix is n, and while accessing arrays element of a two-D array we have to specify the indices for both the dimension.

(Refer Slide Time: 17:33)

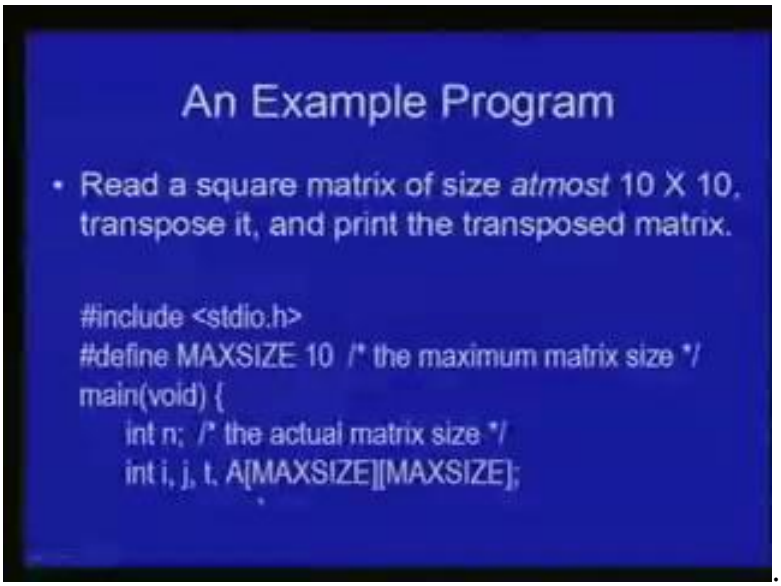


So, just like a matrix element is referred to as a i j if a is declared as a two dimensional array, then we have to access its array elements as A i j in this particular notation. So, we have to give both the indices for the two dimensions, and in the first dimension the value of index gain should range from 0 to the size of that dimension, and this is true for all dimensions.

So, the value of j in this example, assuming that previous declaration where the number of rows was m, and number of columns was n the value of i should be between 0, and m minus 1, and the value of j must be between 0, and n minus 1. So, the indices range from 0 to n minus 1 where n is the size of that particular dimension. So, here are some examples of usage of array elements a i j assigned 5. So, again an expression denoting an array element can occur on the left hand side or on the right hand side.

So, this is initializing the highest i'th element of the array A to the value 5, and here the j'th i'th element of the array B is being assigned to the value of i'th j'th element of another two-D array B. So, this is simple enough. Now, let's look at an example, program using two dimensional arrays. So, let's take a familiar problem from matrices.

(Refer Slide Time: 19:22)

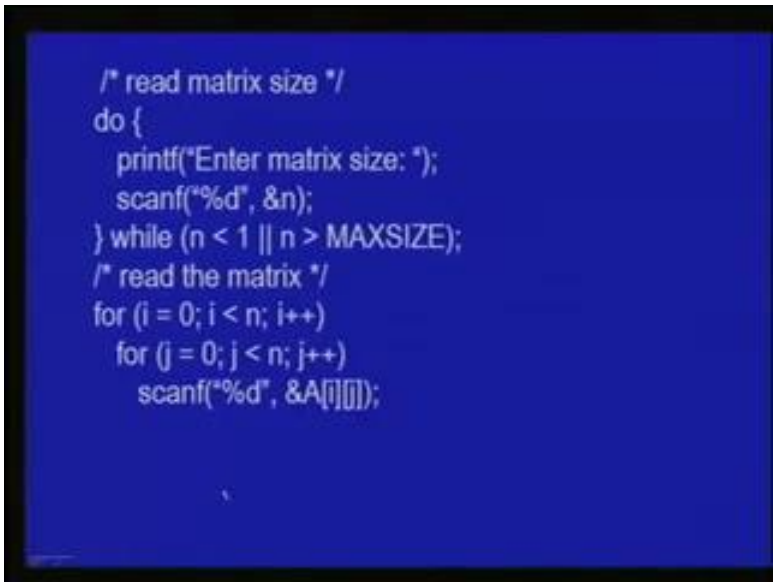


Suppose the problem is to read in take as input a square matrix from the user of the size at most 10 by 10 that is the matrix that user gives may be smaller than 10 by 10, but it cannot be larger than 10 by 10, and this limit has to be placed, because recall that when we declare an array we have to declare the size of the array, and. So, if we declare the size of the array to be 10 we cannot store more than 10 elements in that array. So, that is why we are saying that the a matrix that we read from the user should be of the size at most 10 by 10.

We are supposed to compute the transpose of the matrix, and print the transposed matrix. So, let's start developing this program this is the beginning part we define a symbolic constant representing the maximum size of any dimension of the array, and here are the variable declaration, and it is the actual matrix size that we are going to read from the user. Just remember may be less than ten, but cannot be more than 10 this A is the two dimensional array that we are declaring note that the size of both dimensions is max size which happens to be 10 i, and j, and t will use as temporary variable for looping, and other purposes, and so on.

So, the first part is simple enough the first we have to read the matrix from the user, and for doing that, lets first ask the user for matrix size we have to ask the size in only one dimension, because we are assuming that its always square matrix that we are reading.

(Refer Slide Time: 20:35)



So, this should be familiar this do while loop should be familiar to you from the last lecture, using this two continuously prompts the user for entering the size of the matrix till he gives an appropriate size, and the appropriate size for a matrix will be some thing which is at least one, and at most max size.

So, as long as the value of n is less than one or greater than max size which is 10 we keep giving this prompts to the user to enter the matrix size, and read in the value of n so finally, when we reach here we have read the matrix size n, and it is known it is directed to be between one, and 10, and finally, we have to read a matrix.

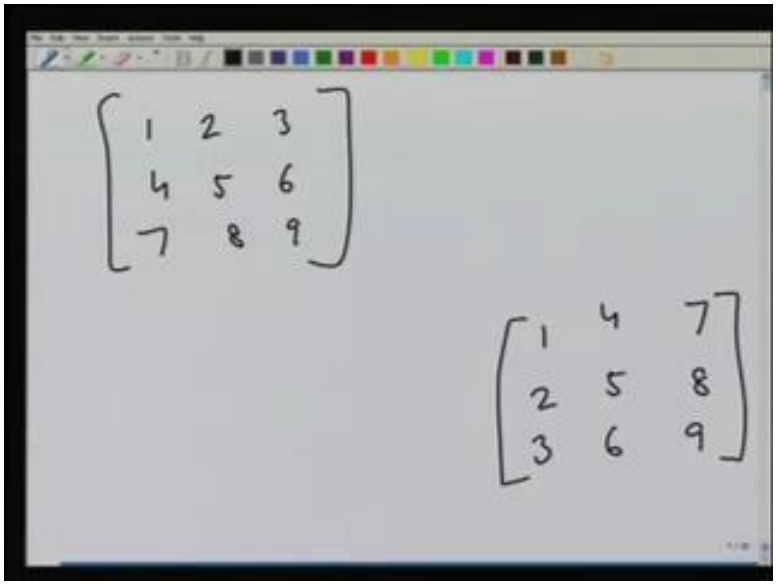
So, for reading a matrix we have to read each element a i j of the matrix in the usual matrix notation we write rows we write matrix row-wise and. So, therefore, we will expect input in the same fashion and. So, for doing that we have used two nested for loops here.

The first for loop transform i is equal to 0 to n minus one note that n is the actual size of the matrix. So, the number of rows is n, and number of columns is also n and. So, the row indices were used will be 0 to n minus 1, and the columns indices that were used will also be from 0 to n minus 1, and this kind of a nested lops structure is very common for manipulating structures like matrices, because this allows us to individually process each element of the matrix 1 by 1.

So, this outer loop runs for every row of the matrix, and for every row this inner loop accesses elements in every column of the matrix, and together these loops will ensure that we access, and process each element of the matrix one by one in the row order, and for each element of the matrix right now, all we need to do is read its value. So, we use a scanf statement to read the value of the matrix.

Let us before writing the program further lets think little bit about how we are going to transpose a matrix. So, let's assume we have a matrix looking something like this the transpose of the matrix is the matrix which has the rows, and columns interchanged.

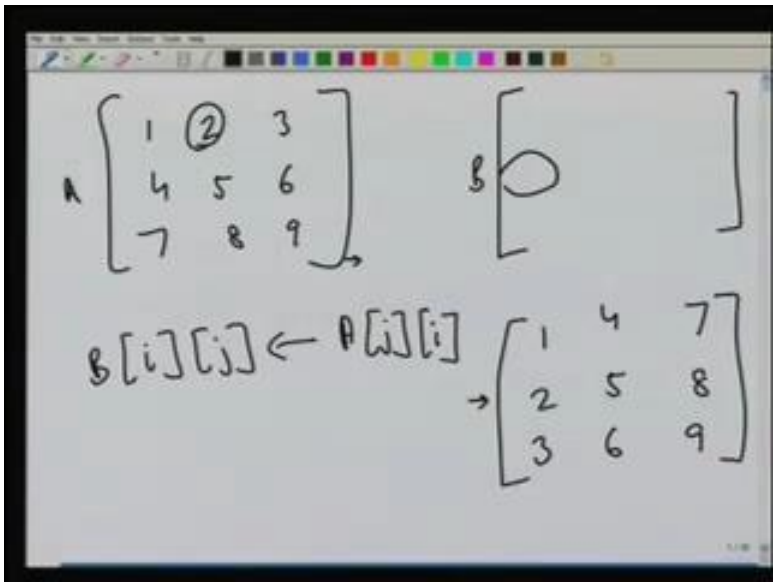
(Refer Slide Time: 23:42)



So, the transpose of this matrix will be this matrix. So, the question is how we are going to convert this array into this array one simple answer could be that we use another array variable of the same size, and assign values in the various elements of this array such that the value at a particular element is the value of the element at the mirror image with respect to diagonal of that element in the original array that is this value should be replaced at this position in the final things.

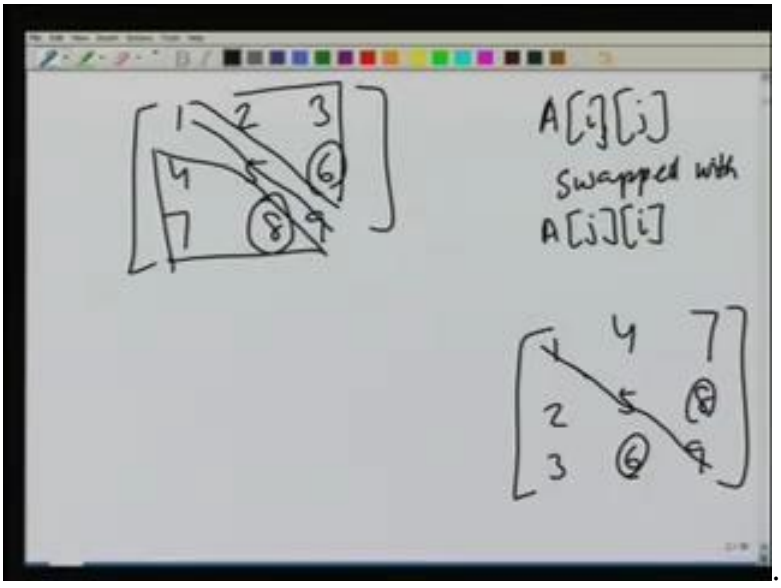
So that means, that if this new array is called B, and the original array was called A, then we should set B i j to A j i. So, that is one way of doing this, but suppose you want to do in different fashion, and we want to transform the original matrix itself. So, that it becomes transpose of itself, then we need to do something different.

(Refer Slide Time: 24:31)



Let us we write the matrix again, and let's right the transpose again too. So, let's look at these two matrices, if you note the diagonal remains the same in the transpose, and if you take any element other than the diagonal, then that gets interchanged with the mirror image of that element on the other side of the diagonal.

(Refer Slide Time: 26:00)



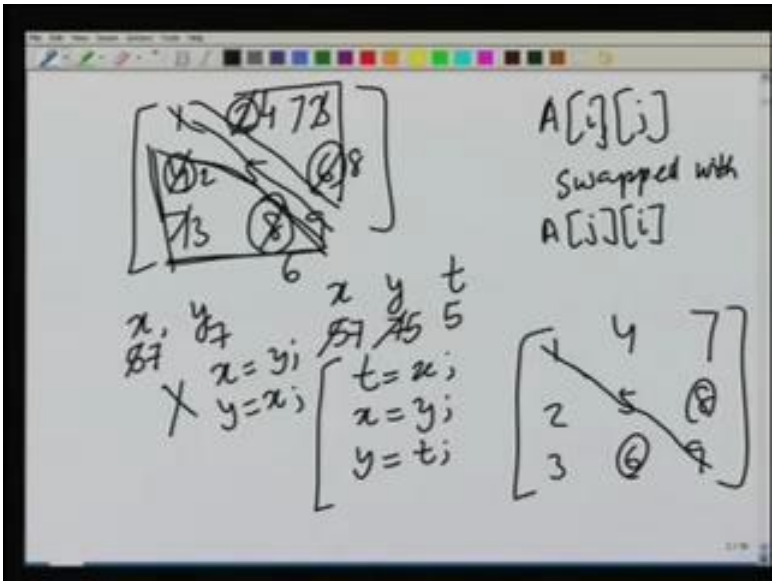
So, we had 6, and 8 here, and we have 8, and 6here so that means, what we need to do is that for all elements on one side of diagonal let's say this triangle or this triangle the lower triangle or the high upper triangle for each element in let us say the lower triangle

suppose we swap with the corresponding element in the upper triangle that is A i j is swapped with a j i. If that happens let's say for this, and this will become 4, and this will become two, and similarly this will become 3 this will become seven this will become eight, and this will become 6.

So, what we need to do for implementing this algorithm is to write a loop which accesses all elements in the lower triangle of the matrix, and for the each element in this lower triangle it swaps it with the mirror image of that element in the upper triangular um upper triangle of the matrix. The elements on the diagonal need not be modified at all, because they remain at their original positions in the final matrix, and the one final thing that we need to do is to find out how we swap two array elements, and the procedure for that is very simple.

Suppose we have to swap two variable value values of two variables x, and y, then clearly swapping like this will not really work this will not work, because let's say x had the value 5, and y had the value 7.

(Refer Slide Time: 28:11)

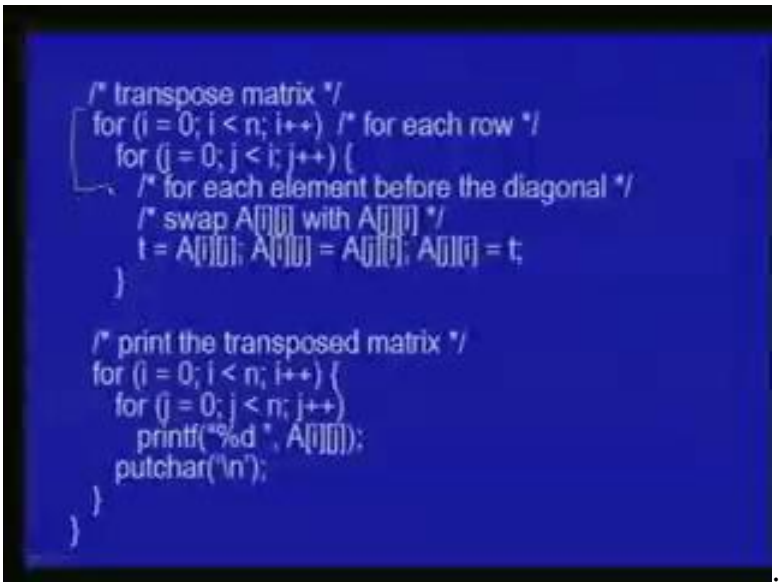


When x is assigned the value y it become seven, and the original value of x is lost. So, when we do when we do the assignment y assigned x the value of y remains seven instead of doing this what we need to do is to take a temporary variable, and before we assign to x the value of y we have to store the old value of x into the variable t. So, this is what it could look like. So, when we do this let us see what happens when t is assigned x

the value of t becomes 5 in this example, then x is assigned y. So, x becomes seven, but we have lost not lost your original value of x, because it is still available in t and. So, the value of y is changed to the value of t which now, becomes 5, and the same procedure can be used for swapping the array elements.

So, having understood this let's go on, and see how we can write the program for implementing this algorithm. So, coming back to the program here we get the matrix the matrix. Now, let's transpose this. So, we have loop here, and you will notice that it is a slightly different loop the first outer loop is the same as the before for i from 0 to n minus 1.

(Refer Slide Time: 28:31)



So, this will run once for every row in the matrix, but for every column in the matrix we want to access only the elements in the lower triangle of the matrix. So, the array elements in that row that should be accessed are the ones which lie before the diagonals, and the diagonal element in the row i is at column i. So, therefore, we should only access array elements at columns 0 to i minus 1 and. So, therefore, you can see that this loop runs from 0 to minus one it terminates when j becomes equal to i.

So, for each element in the row before the diagonal we have to swap the i j with A j i. So, use the swapping technique that we just saw t assigned A i j Ai j assigned A j i A j i assigned t. So, this swaps the highest i'th j'th element with the j'th i'th element, and that is the end of the loop note that the inner loop has three simple statements in its body, and

so, therefore, they have been clubbed into a compound statement using these curly braces as far as the outer for loop is concerned the body consists of this single for statement, and therefore, the braces are not required.

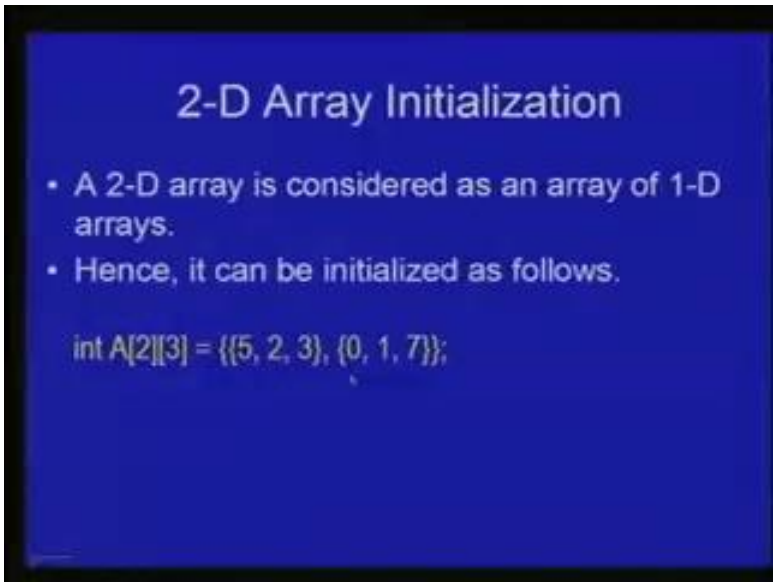
So, finally, when the loop ends its time to obtain transpose matrix, and we just print that print the matrix row wise. So, for each row we print all the elements 1 by 1. So, this loop will print all the elements in the i'th row 1 by 1. Note that after the percent d there is a blank space. So, that after printing one there is a blank space left in the output, and then the next array element is printed and. So, on, and after this entire loop has finished.

So, the first row in the transpose matrix has been printed. So, let's say that was one two three, and then after that you want to start printing the next row on the next line note also that, there was no back slash n in the format string of the first printf. So, that will ensure that all these three values or all n values of the same rows are printed on the same line with a blank in between while at the end of the row we want to go to next line. So, that next row is printed on the next line in the familiar matrix notation.

Therefore once this row has been has been printed using this inner for loop the print one new line character using the familiar putchar function, and note that now, for the outer loop in the body there are two statements. One is this for statement, and one is the statement calling the library putchar and. So, therefore, we have enclosed these statements in braces for the outer loop, and that is the end of the program.

Lets now, see how two dimensional arrays can be initialized that is again very simple the two dimensional array is essentially considered to be an array of one dimensional array that is if we have an array of size 2 cross 3 it is considered to be an array of two arrays of three integers each, and so, it can be initialized like that.

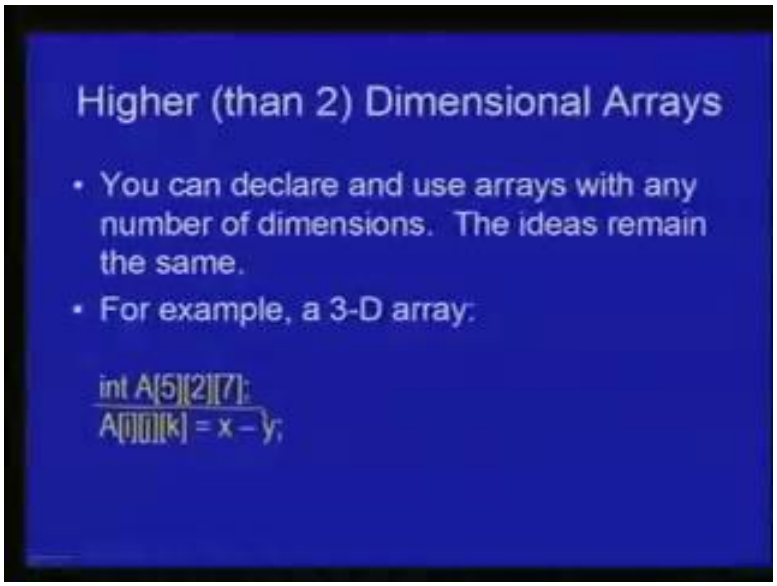
(Refer Slide Time: 31:37)



So, we have to give the value of each row which itself is an array within the curly braces. So, this is row 0, and this is row one within row 0 this is the element at column 0 column 1, and column 2, and so on, and so forth. So, this is surely straight forward next lets quickly, see how we can use arrays with dimension higher than two or more than two, and the ideas are very similar to what we have already seen for two dimensional arrays the natural generalization hold.

So, we can declare, and use arrays with any number of dimensions in very similar ways to what we have already seen for example, if 3 D arrays could be initial could be declared like this giving the sizes on all the 3 dimension, and when we access an element of the array, we again have to give the indices at all the three array dimension.

(Refer Slide Time: 32:47)



.

So, this is the end of today's lecture, in the next lecture, we will be talking about some special kinds of arrays which are known as strings, and which are very useful in manipulating data, which are which comprise of text rather than numbers. So, we will look at strings in the next lecture.