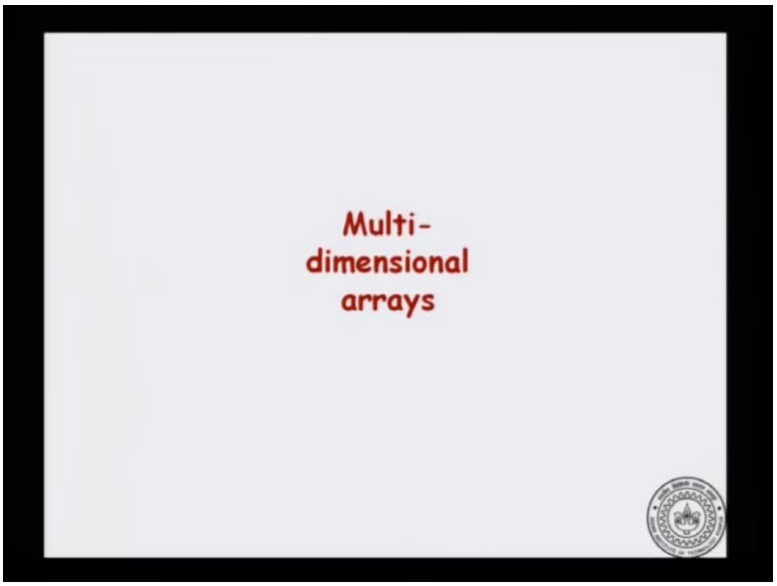
Introduction to Programming in C
Department of Computer Science and Engineering

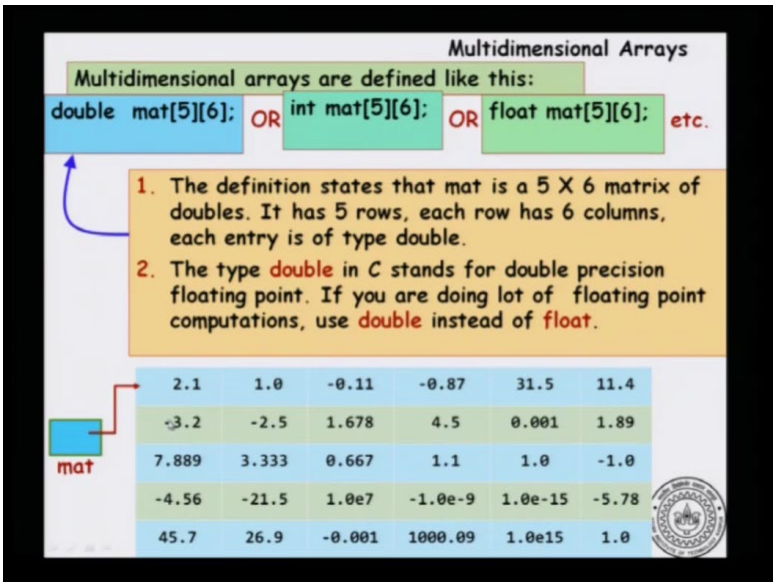
In this video will look at multi-dimensional arrays.

(Refer Slide Time: 00:03)



In particular, let us look at two dimensional arrays. Because, that will give you an idea how multi-dimensional arrays work. Initially, let us look at them as arrays and in a subsequent video will look at the connection between multi-dimensional arrays and pointers.

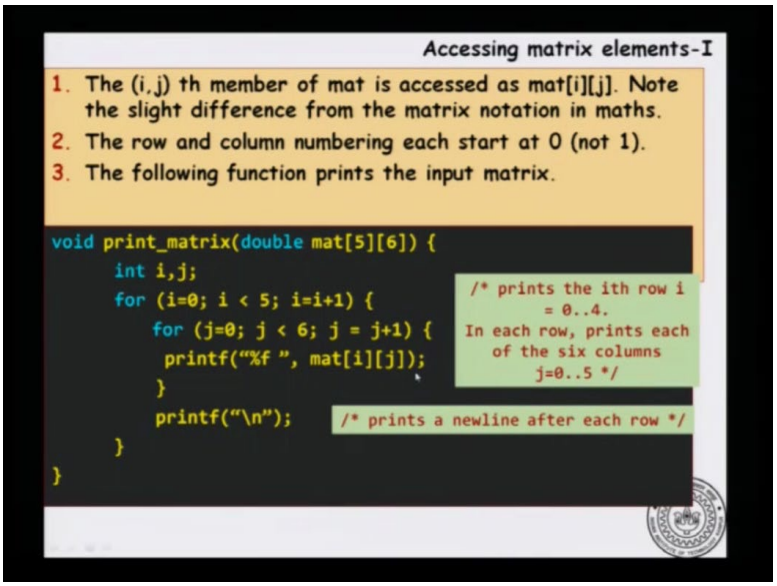
(Refer Slide Time: 00:22)



So, multidimensional arrays can be defined in the similar to the following, you can say double mat[5][6] or int mat[5][6] or float mat[5][6], this is similar to the mathematical notation of multidimensional arrays are matrixes. So, let us look at the first example, we have that the definition states that mat is a 5×6 array of double entries. So, this means that mat has 5 rows, each row contains 6 entries and all the entries are of type double. Double is what is known as double precision floating point numbers.

And if you are doing a lot of floating point computations, then instead of float you could use double because, you might need a lot of precision in your computation. So, the matrix 2D array might look like this, this looks like a mathematical matrix of size 5×6. So, it has 5 rows, rows 0 through row 4 and each row has 6 columns, column 0 through column 5.

(Refer Slide Time: 01:46)



Now, the i j th member of matrix is accessed as mat[i][j] this is slightly different from the mathematical notation. In mathematical notation you will write matrix and then a square bracket and then you will write i comma j followed by close bracket. So, this is different in c, you would write the indices separately in their own square brackets. Now, the row and the column numbering begin at 0, this is similar to one dimensional arrays we saw that one dimensional arrays start with index 0.

Let us look at a function which prints the input matrix. So, I have a function it returns void. So, it does not return anything it just performs an action, which is to print a matrix. Now, the function is called print matrix, it takes a double matrix mat of size 5×6, 5 rows 6 columns each. I first declare i and j, i is suppose to iterate over the rows and j is suppose to iterate over the columns. Now, how do you iterate over the whole matrix.

Well, first you would take each row i. So, you need an i outer loop for that based on the variable i. I will go from 0 to 4, so the for loop goes from 0 until you hit 5. Now, for each row what do we have to do, we have to take the elements in the column. Now, the columns are numbered 0 through 5. So, for each i th through we have to take column 0, column 1, column 2, column 3, column 4 and column 5. So, all these entries and then you have to print that entry.

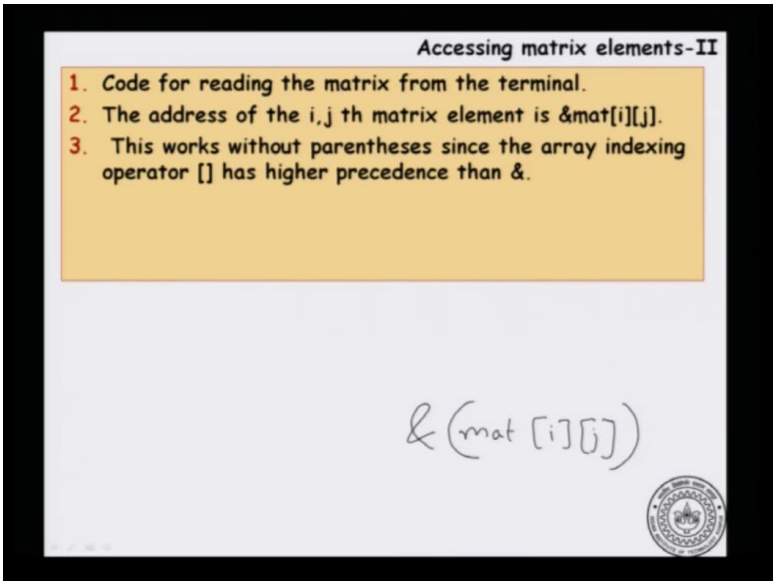
We have just mention that the i j th element in the matrix is accessed as math i j, i in square bracket and j in square bracket. Therefore, you will say printf("%f", mat[i][j]). So,

this will take the entry in the i th row, j th column. One more thing that is worth noting is that, even though you had a double matrix, you still print it as %f as though you were printing a float and the language will take care of printing the double precision.

So, here is the loop to print the columns of a row. Once you are finished with the row, you would a print a new line. Because, then you can start at the beginning of the next line for the next row, so here is the loop. So, what the loop does is prints the i th row, row starting from 0 and ending in 4 and for each row print each of the 6 columns 0 through 5. Now, at the end of the each row you would print a new line. So, here is the code to print a matrix, the lesson here is how to access the i j th element. You would access it as mat[i][j].

Now, the dual operation of printing is of course, reading in the input from the user, we have done it using scanf. So, let us try to use scanf to read in elements which are input by the user.

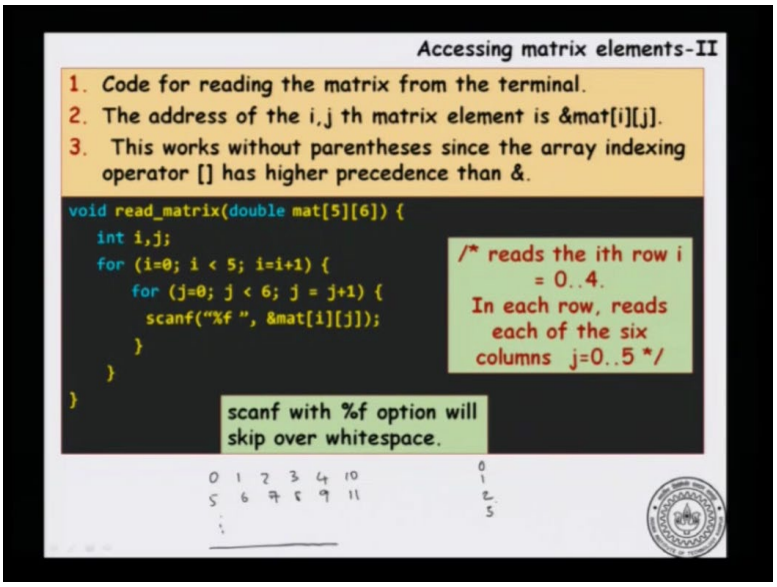
(Refer Slide Time: 05:29)



Now, one of the things with the scanf is that the argument to which variable we have to read it in, we usually give and x, if you have to read it into a particular variable x when we will say scanf whatever format it is and then say and x, which says the address of x. Similarly, I could guess that in order to read to the i j th element of a matrix, I would need &mat[i][j] and that is actually correct, you do not need a parentheses here to right.

So, you do not need to right &(mat[i][j]) with mat[i][j] in square bracket. Because, the address operator has lower precedence then the array indexing operator. So, when you see an expression like this, the parentheses will be in such a way that mat[i][j] is evaluated first and then the address is taken. Because, it is a lower precedence operator. So, this is similar to a one dimensional array that you have already seen, you would have read it using & of a i. So, we have the two dimensional analog of that.

(Refer Slide Time: 06:44)



So, let us look at the code and the code looks exactly as the print routine, except that we are now scanning a number. So, you have an outer loop which will go through all the rows and then in inner loop which will go through the columns of the i th row and how do you scan. You says scanf("%f", &mat[i][j]). So, remember if it was just a double variable instead of an array, you would have just said & of the variable name. Similarly, we have &mat[i][j].

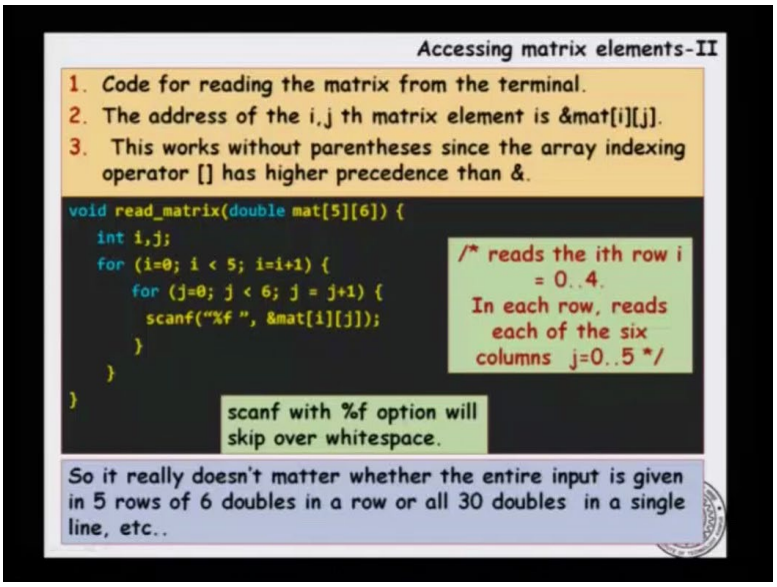
Again, note that even though we have a double array, you read it in exactly as though it were a float array, using %f format. So, read in the i th row and i th row goes from 0 through 4 and for each row read the j th column, column goes from 0 to 5. Now, the way this scanf works, the scanf with %f option will skip over the white space and it will skip over any white space.

So, in effect what it means is that, if I had to enter a matrix of size 5×6, I can enter it in multiple ways, I can enter it in the most natural way which is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

and so on. So, 5 rows each row has is entered in a line and each line has 6 entries, so let us call this may be 10 and 11. So, each row has 6 entries and there are 5 rows, this is a most natural way to enter it.

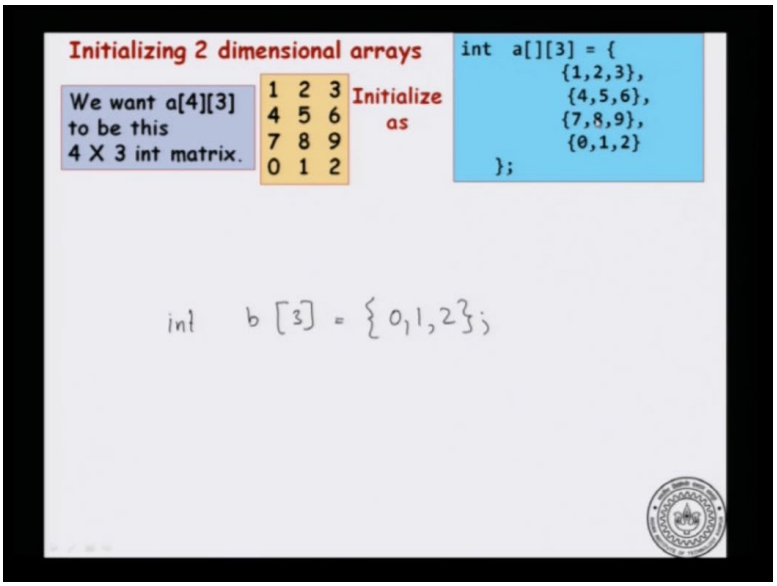
But, as for as scanf is concerned any white space will be skip. So, instead I could just enter one number in one line. So, I could enter it one number per line and it put be read exactly in the same manner, that is a property of scanf.

(Refer Slide Time: 08:53)



So, it really does not matter whether the entire input is given in 5 rows of 6 doubles or just 30 doubles each number in a single row by itself. So, that is you both of them are fine. We have seen how to print an array. We have seen how to read elements into an array. Now, let see how to initialize a multi-dimensional array.

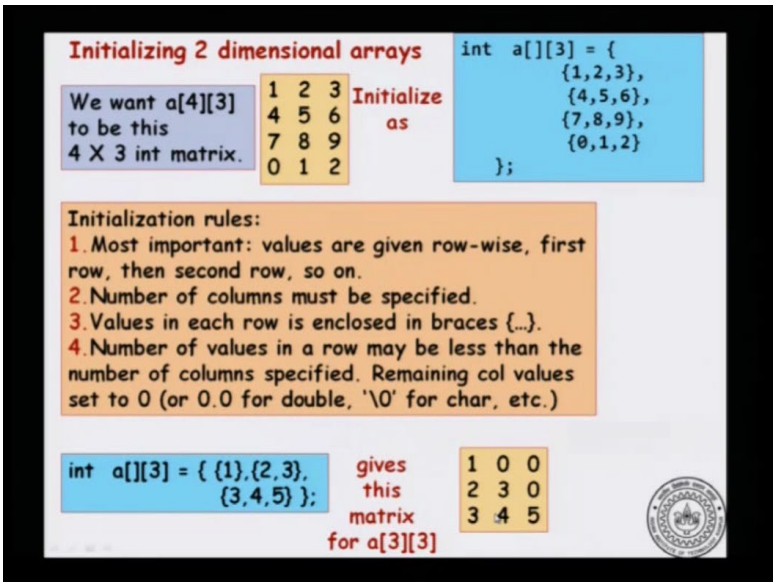
(Refer Slide Time: 09:17)



So, we want to initialize let us a 4 by 3 array in the following way, it should be 1, 2, 3, 4, 5, 6, 7, 8, 9 and 0, 1, 2, let say this is the array that I want to enter. Now, we have seen initialization of one dimensional arrays, if I let say int b 3 how did we initialized, we could initialized it us 0, 1, 2. So, we summary of this is that, it is a list of numbers separated by commas and the list is enclosed in curly braces, this is the case for a one dimensional array.

So, it is natural to generalized the notation in the following way, if I have to initialize a 4 by 3 array, I can just say curly brace. And so here is a list of elements and each element is basically a row. So, what is a number here will be a row? So, it will be a list of rows and each row being an somewhat like an array, each row will be given by a list. So, the array initialization on the right hand side is exactly the array that is shown here. So, it will come out to 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2. So, the notation is consistent and it is a generalization of the one dimensional array notation.

(Refer Slide Time: 10:51)

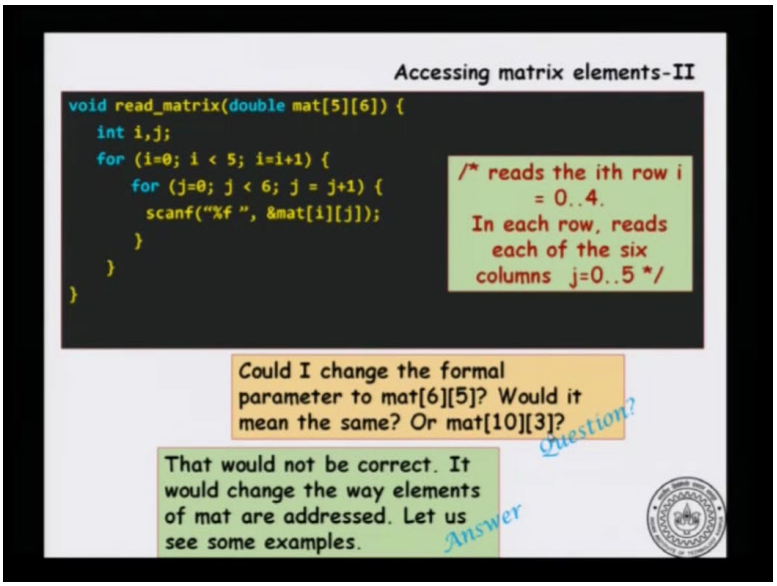


So, there are some initialization rules, similar to what we are seen for one dimensional arrays, values are given row wise. The row number 0 is the first entry, number of columns needs to be specified, we need to know how many columns there are? Now, value of each row is enclosed in {} and the number of values in a row, may be less than the total number of columns, this is allowed.

This was similar to how we saw that, even though you had declared the size of an array, you could give one dimensional array, you could be less than that number of values as the initial values. The remaining values will just be 0, same case occurs in the multidimensional array. So, let us watch an example, if I have an array a number of rows unspecified, number of columns 3.

But, each row let say I have 3 rows, each row does not have exactly 3 elements, one the row 0 has just 1 element, row 1 has only 2 elements and so on, it will be initialized as 1 0 0 because in row 0 I have given only 1 element. So, that will be the first and the remaining will be 0, 2 3. So, I am short of 1 element that would be 0, 3 4 5 I have 3 columns and I have given 3 values. So, it will be initialized as 3 4 5. So, the initialization on the left hand side results in the matrix on the right hand side, here is all initialization words.

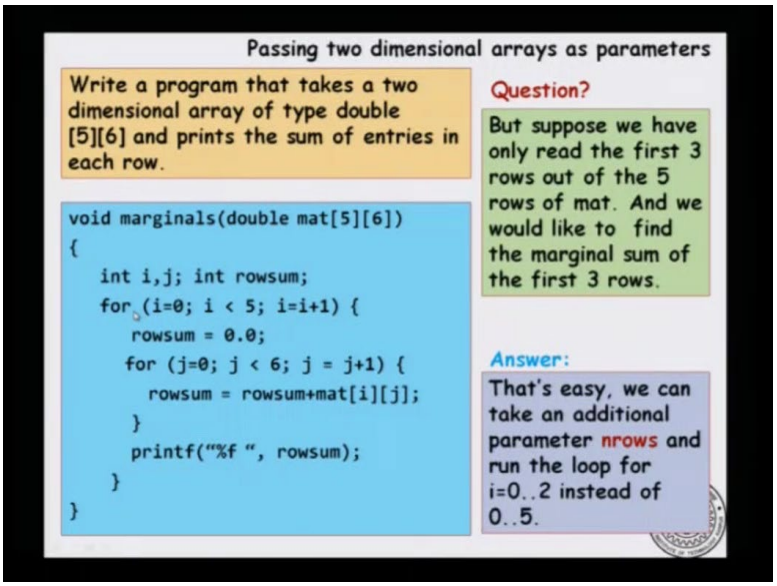
(Refer Slide Time: 12:40)



Now, let us look it access mechanism in somewhat greater detail. So, let us ask the following question, instead of matrix 5×6 I have return and function to read a matrix of size 5×6, can I give a 6 by 5 matrix? So, this is a matrix of 5 rows, 6 columns each instead can I give a matrix of 6 rows 5 columns each, the total number of elements is till 30 would it be the same or would it be even a matrix of [10][3], 10 rows 3 columns each all of these have 30 elements.

Now, as for as c is concern are all these the same, the answer is that no, it is not correct neither it should be. But, we will say that the answer depends on the way the array elements are accessed. So, will see this in greater detail.

(Refer Slide Time: 13:40)



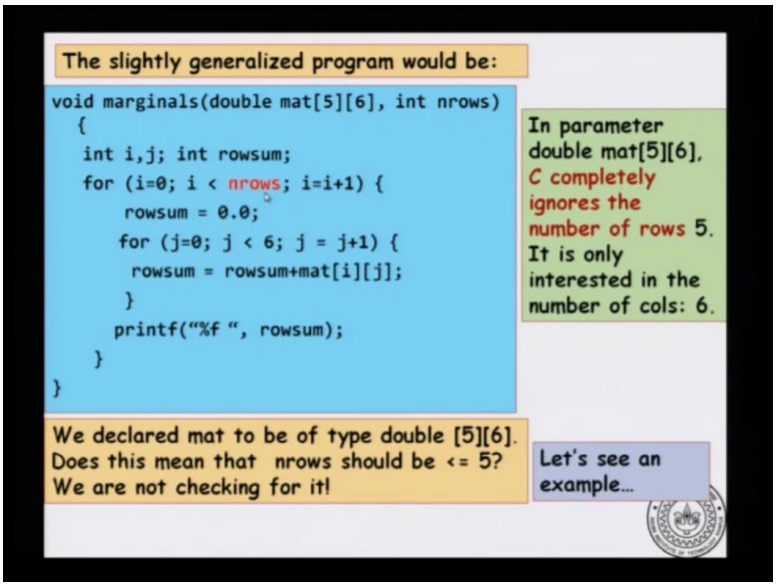
So, in order to motivate there let us introduce the problem of passing an array to a function and let us look at the issue in greater detail. Suppose, I want to take two dimensional array of type double [5][6] and print the sum of entries in each row. So, this is similar to a matrix program, that we have seen much, much before given a 2D matrix, for each row you have to find the sum of elements in each row and just print it out.

So, in mathematics this is often called marginal's. So, let us just compute the marginal's, we have a function void marginal's, it takes matrix [5][6], it has int i j, i is over the rows, j is over the columns and I also have a row sum variable to keep track of the sum of a row. So, what do I do, I have an outer loop which goes through all the rows, for each row I initialize the sum to 0. Now, for each row I have to some all the elements in the i th row. So, I have to sum all the elements in the columns j 0 through 5 of matrix i j.

So, I will go through the elements and add them into the row sum. Once I am done with the last column of row i, I have the row sum for row i and I will print that. So, this printf is happening in the loop for row height. Now, let us look at a slight modification, we say that instead of printing 5 rows I currently have only 3 rows of entries available. So, can you print me the row sum of the first 3 rows, instead of all the 5 rows.

Now, this is very simple let us just modify the function a little bit, it takes an additional parameter saying, how many of the initial rows do you want me to sum? So, that is an additional parameter, let us call it n rows.

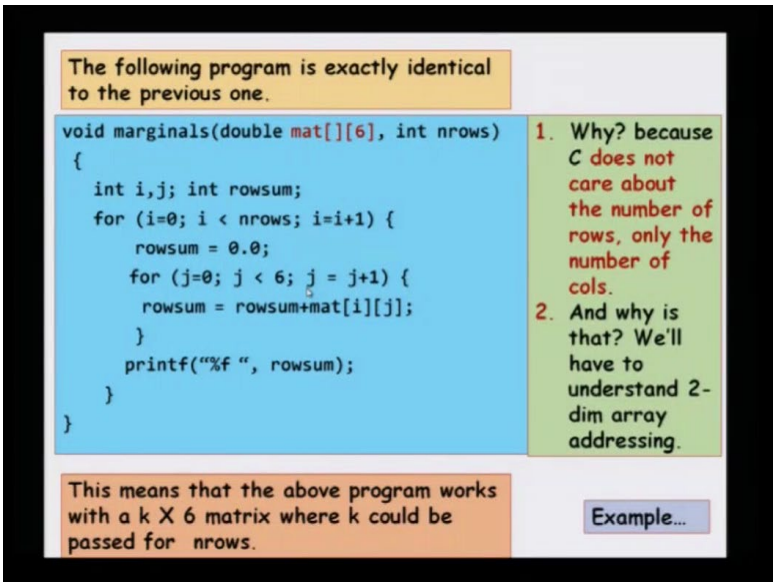
(Refer Slide Time: 15:54)



So, here are the number of rows for which I have to take this sum. And that function is a very small modification of the function that we have already seen. The difference is that, we now take n rows which is like, how many rows do we have to add and then for i = 0, earlier I would go from i = 0 to 5. Because, the matrix had 5 rows, but in now I will just say I will go up to n rows and the logic is the same as before, nothing else changes.

So, his strange things he completely ignores the number of columns, for as for as the c languages concern, if you have a 2D array, the number of columns is crucially it has to be specified. But, the number of rows is not really important. So, c completely ignores the 5 part, the number of rows. Now, this means that we could pause less than 5 rows into the same function. Since, we are not checking for example, that encloses <= 5.

(Refer Slide Time: 17:11)

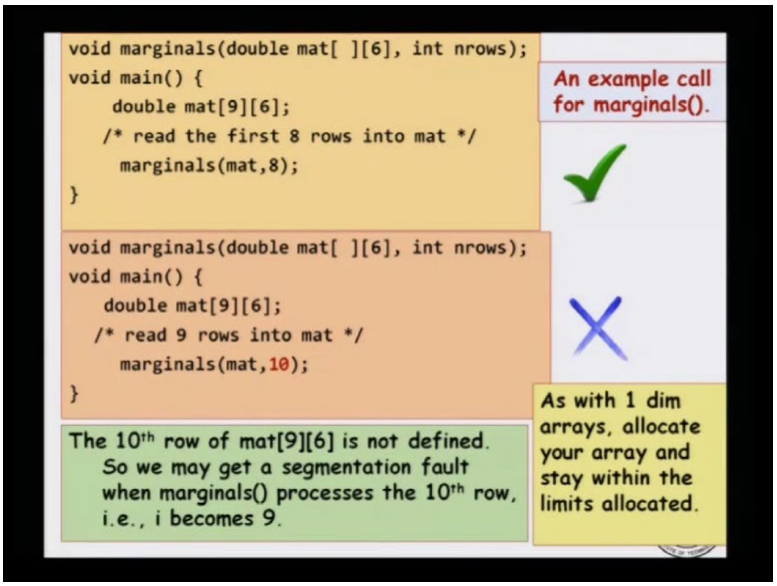


So, let see an example here is the completely surprising example, that this code is the same as before, though only difference is highlighted in read, that I have now omitted what is the number of rows? Please relate this back to codes, that we used to right for arrays. Earlier, I said that for an array, you do not need to specify the number of elements in the array, when you write a function taking an array as parameter I could just say int arr and then empty pair of [ ] with no size in between.

So, we have a similar phenomenon for 2D arrays, except you are not allowed to omit both rows and columns, you have to specify the number of columns. But, you have the flexibility that you are allow to omit the number of rows. So, the number of rows is not important, you could omit it and just given empty pair of brackets and the code will work as before.

So, this means that the above program actually works for any k × 6 matrix, where k could be the number of rows. And this is because c does not care about the number of rows, only about the number of columns and y is this asymmetry, why is said that it case about the number of rows, but not the number of columns, will see this using the two dimensional array addressing.

(Refer Slide Time: 18:43)



Let say that I have return code for computing marginal's and it takes these parameters double mats empty pair. So, the number of rows is un specify, the number of columns is 6 and then it takes an additional parameter n rows, which says how many rows do should I add. And then I am calling this function, suppose I have define the function elsewhere and I am calling this function from name. So, I declare a matrix 9 by 6 and then I will call marginal's on just the first 8 rows not the 9th row.

So, I passes subset of the rows, this is 5. Because, I have declared as matrix of size 9 by 8, but I am passing only 8 rows to marginal's and that is fine, I can passes subset of the rows. What is definitely not fine is, suppose you declare a matrix of size 9 by 6 and say that I want you to find the marginal's of the first 10 rows, then this is unsafe. Because, it is true that the marginal's function does not really care about the number of rows.

So, it will work for any k × 6 matrix. But, you cannot hope to pass arbitrary junk values to that matrix. For example, you have just declared a [9][6] matrix. Now, the 10th row of the matrix is basically invalid. So, if you pass it you could expect your code to receive a segmentation evaluation, when your run the code. So, when it processes the 10th row what it, it was basically cross the limits of the array. So, the code may have a segmentation evaluation.

So, note the difference between saying that it could marginal's could work with arbitrary k × 6 matrices, it is till true that if you pass junk values to the matrix, your code will

crash. If your code is a valid matrix, then you can pass an arbitrary number of rows inside the matrix. So, the summary is that as with one dimensional arrays, you should allocate your array and stay within the limits allocated, within those limits the number of rows does not matter. But, it does not mean that you can over suit the limit and hope that your code will work, it may not and it can crush.