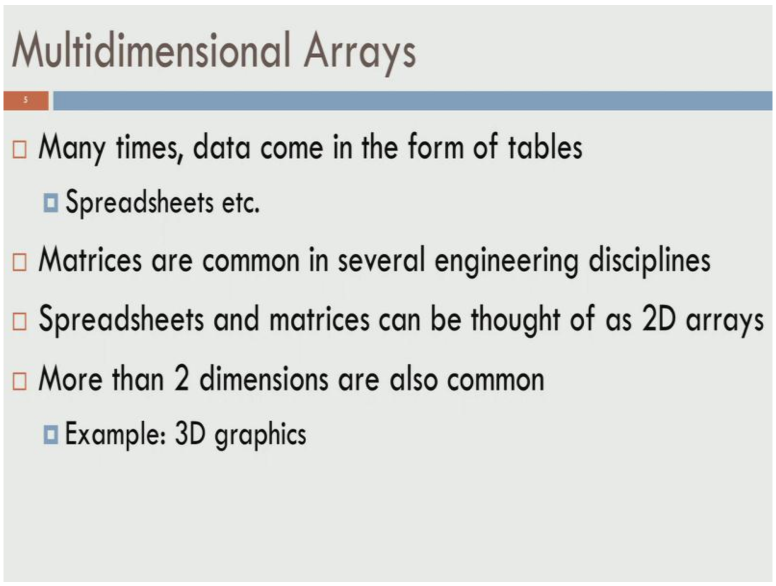
Programming, Data Structures and Algorithms

Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute Technology, Madras

Module - 9A
Lecture - 13
Content
Multi-dimensional arrays: two-dimensional and more
Matrices,matrix operations

Welcome to lecture 4. We have a bunch of modules, which are all related to arrays and what are called pointers. So, the first module is about multidimensional arrays.

(Refer slide Time: 00:26)

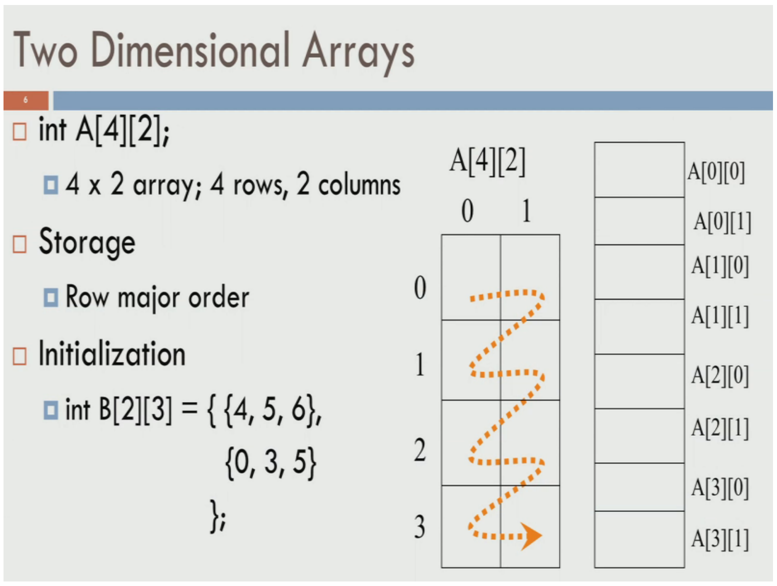


Multidimensional Arrays

- Many times, data come in the form of tables
  - Spreadsheets etc.
- Matrices are common in several engineering disciplines
- Spreadsheets and matrices can be thought of as 2D arrays
- More than 2 dimensions are also common
  - Example: 3D graphics

So, multidimensional arrays – they appear many times in the form of tables. So, I am sure you have seen spreadsheets or matrices and so on. These are all very common things in several engineering disciplines. And you can think of them as two-dimensional array. So, you have one dimension, which is the set of rows; another dimension which is the set of columns. And you can have rows and columns, which form a table. This could be a 2D array. You could also have arrays, which are more than 2 dimensions. For instance, if you look at graphics, 3D graphics will require x, y and z. So, you have 3 dimensions and so on. So, we need a mechanism by which we can not only store, but also be able to access them and manipulate them as variables and so on.

(Refer Slide Time: 01:19)



Two Dimensional Arrays

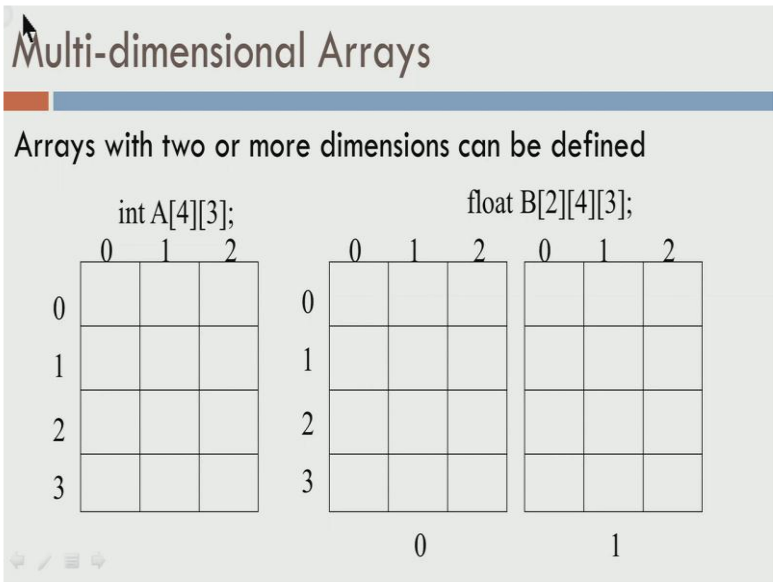
- int A[4][2];
  - 4 x 2 array; 4 rows, 2 columns
- Storage
  - Row major order
- Initialization
  - int B[2][3] = { {4, 5, 6},
    {0, 3, 5}
    };

So, let us look at what a 2D array would be. So, in this example, what we have is we have an array called A, which says int A of 4 comma 2. So, that is the declaration. So, just like what we have for basic variables, int A of 4, 2 tells you that, you want 8 integers arranged as 4 rows and 2 columns. And as with 1D array, one thing that you will have to remember is that, you have 4 rows numbered 0, 1, 2 and 3; and 2 columns numbered 0 and 1. So, the rows get numbered from 0 and the columns get numbered from 0 as well. This is something that you have to remember. This has to be drilled into you. And the storage in the memory is actually what is called row major order. So, what I mean by that is as follows. So, even though you have 2 dimensions as a pictorial representation here, remember – memory is just a sequence of memory locations; it is actually 1-dimensional.

And the way things are going to be stored is as follows. You take the first row, that is, the 0-th and you take the first column, that is, the 0-th column. The entry A of 0, 0 goes into one location. Then A of 0 comma 1, which is the next column in the same row, gets into the next location. And then you can think of this as folding down. And you have the first row and the 0-th column that goes into the next location and so on. So, you can see that, the order in which things are listed are… You have start with 0, 0; then you have 0, 1;

and then you have 1, 0 and 1, 1 and so on up till 3, 1. So, you have 8 memory locations of the type integer and they are all contiguous. So, this is called row major order. You can initialize 2D arrays as you did for 1D arrays also. So, for example, here we have a 2D array called B, which has 2 rows and 3 columns. So, this set of set braces tell you that, you want an array. And within this, you have two such curly braces – two sets of curly braces and list of values there. So, the 0-th row will contain 4, 5 and 6; and the 1-th row will contain 0, 3 and 5. So, we have two different things and the entries are again going to be laid out in row major order.

(Refer Slide Time: 03:51)

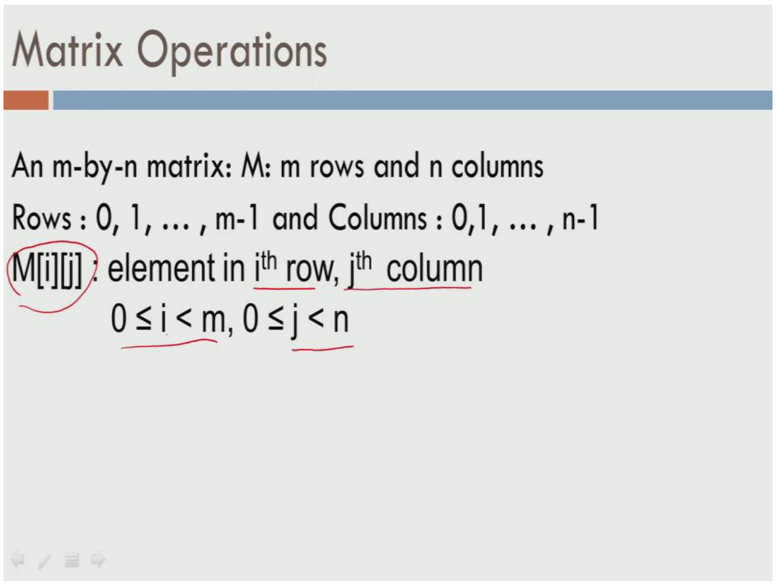


Multi-dimensional Arrays

Arrays with two or more dimensions can be defined

You can go and design or use arrays, which are more than 2 dimensions. So, for example, here we have float B of 2, 4, 3. So, you can now think of this as two planes; each plane having 4 rows and 3 columns. So, you have the 0-th plane and the 1-th plane. So, I will call it 1-th just, so that I can say i-th later. This is the 0-th plane and the 1-th plane. And within that, I have 0-th row, 1-th row, 2-th row and 3-th row. And similarly, I have 0-th column, 1-th column and so on. So, essentially, you can access i-th row of this plane by accessing B of 0 comma i. And if I want to access i-th row and j-th column of this plane, I can say B of 0 comma i comma j. If I want to access something in here, I have to use B of 1 comma i comma j and so on. And see you can actually have dimensions more than three also; but for most practical purposes, you will need 1D, 2D, and 3D arrays.

(Refer Slide Time: 05:09)



Matrix Operations

An m-by-n matrix: M: m rows and n columns

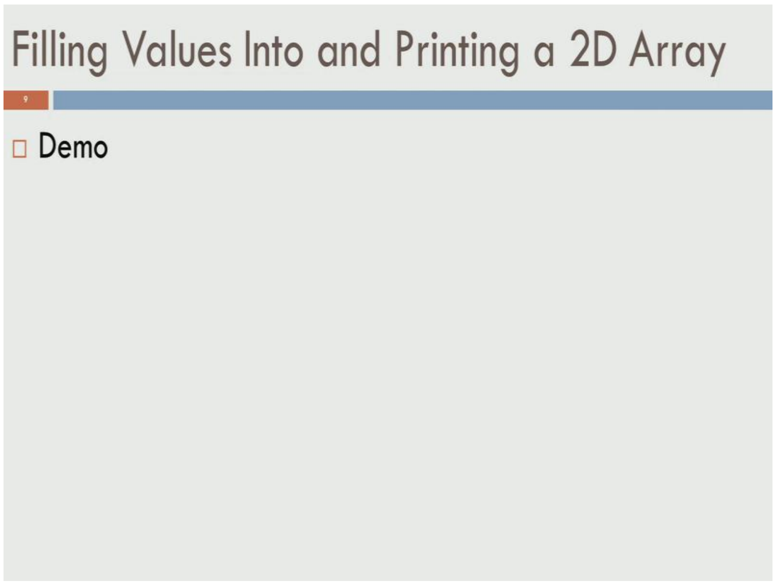
Rows : 0, 1, … , m-1 and Columns : 0,1, … , n-1

M[i][j] : element in i^{th} row, j^{th} column

$0 \leq i < m, 0 \leq j < n$

So, let us look at the most common use of 2D arrays. These are usually for matrices. Let us look at a matrix called M; let it have m rows and n columns. So, the rows are going to be numbered from 0 to m minus 1; and columns are going to be numbered from 0 to n minus 1. So, you have m rows and n columns. And if you want to access the i-th row and j-th column – element in the i-th row and j-th column, you access it as M of i comma j. So, m of i comma j is i-th row and j-th column. Just as you have for arrays, i should be less than m and it starts from 0; j should be less than n and it starts from 0 as well. So, this is just an extension of what you did for 1D arrays.

(Refer Slide Time: 06:05)

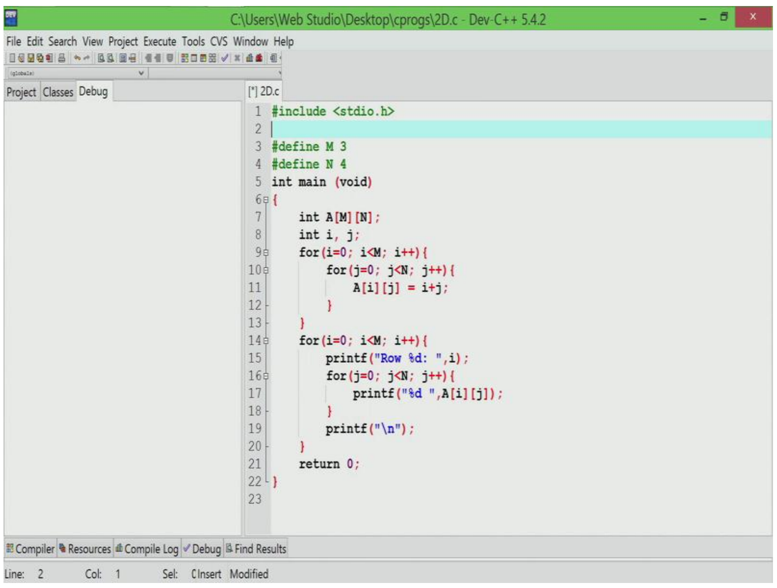


Filling Values Into and Printing a 2D Array

- Demo

So, let us write a small program in which you are going to fill up values and print a 2D array just to show you how access to an array works.

(Refer Slide Time: 06:18)



C:\Users\Web Studio\Desktop\cprogs\2D.c - Dev-C++ 5.4.2

File Edit Search View Project Execute Tools CVS Window Help
Project Classes Debug
[*] 2D.c
Compiler Resources Compile Log Debug Find Results
Line: 2 Col: 1 Sel: 0 Insert Modified

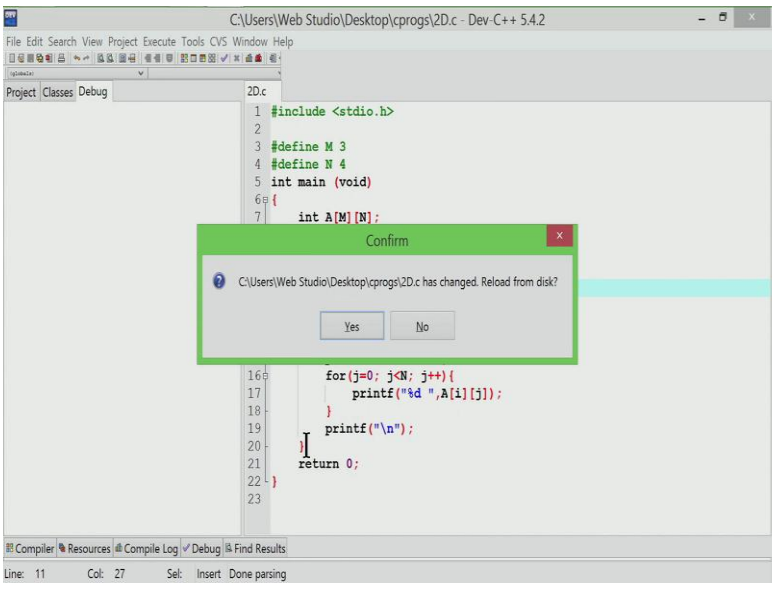
```c
#include <stdio.h>

#define M 3
#define N 4
int main (void)
{
    int A[M][N];
    int i, j;
    for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            A[i][j] = i+j;
        }
    }
    for(i=0; i<M; i++){
        printf("Row %d: ",i);
        for(j=0; j<N; j++){
            printf("%d ",A[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

So, I have this small program written up. So, let us look at this program here. So, I have M, which is defined to be 3 and N, which is defined to be 4. And what I have is I have int

A of M comma N. So, this is going to allocate an array A of 3 rows by 4 columns. So, total of 12 elements. And you can access an individual element using A of i comma j. So, we have this loop running from 9 to 13, which is going to fill up the array; and the loop running from line number 14 to 20, which is going to print the array. So, in this case, these array is the 2-dimensional array is being filled up in this form. So, A of i comma j is going to have the value i plus j. So, A of 0, 0 will have 0; 0, 1 will have 1; 0, 2 will have 2 and so on. So, the 0-th row will have 0, 1, 2, 3 as its elements. Row number 1 will have 1, 2, 3, 4 as it is elements and so on. And we want to be able to print it. So, that is there in this loop. You can see that, it is a nested for loop that is used here. So, this loop starting at line number 9, ending at 13 is the outer loop; and this loop is iterating over the rows. And this is the inner loop and that is iterating over the columns. And this you are accessing A of i comma j. So, you are accessing each element in a row and you go to the next row and so on. And you have the same order in which you are printing this.

(Refer Slide Time: 08:19)


C:\Users\Web Studio\Desktop\cprogs\2D.c - Dev-C++ 5.4.2
File Edit Search View Project Execute Tools CVS Window Help
Project Classes Debug
2D.c

```
1  #include <stdio.h>
2
3  #define M 3
4  #define N 4
5  int main (void)
6  {
7      int A[M][N];
```

Confirm

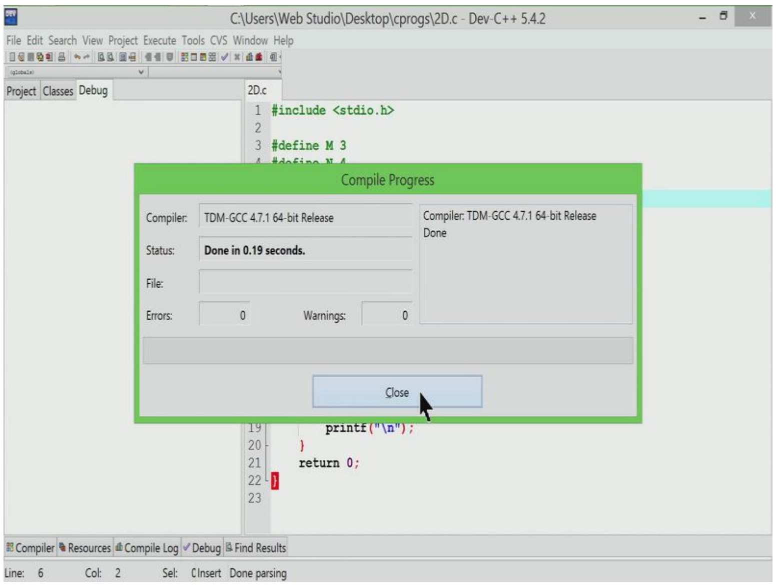
C:\Users\Web Studio\Desktop\cprogs\2D.c has changed. Reload from disk?

```
16         for(j=0; j<N; j++){
17             printf("%d ",A[i][j]);
18         }
19         printf("\n");
20     }
21     return 0;
22 }
23
```

Compiler Resources Compile Log Debug Find Results
Line: 11 Col: 27 Sel: Insert Done parsing

So, let us save this and compile and run it.

(Refer Slide Time: 08:26)



C:\Users\Web Studio\Desktop\cprogs\2D.c - Dev-C++ 5.4.2

```c
#include <stdio.h>

#define M 3
```

```c
        printf("\n");
    }
    return 0;
}
```

Compile Progress

Compiler: TDM-GCC 4.7.1 64-bit Release

Status: Done in 0.19 seconds.

File:

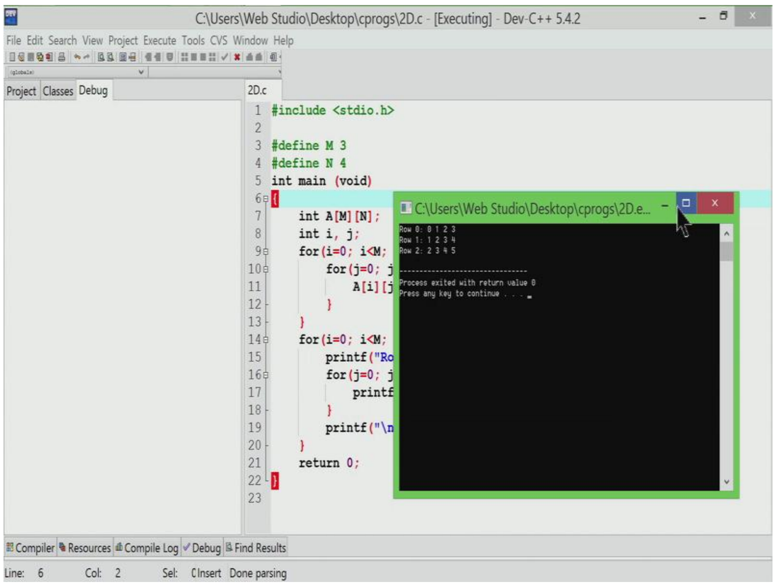
Errors: 0 Warnings: 0

Compiler: TDM-GCC 4.7.1 64-bit Release
Done

Line: 6 Col: 2 Sel: 0 Insert Done parsing

So, I am going to run that.

(Refer Slide Time: 08:29)



C:\Users\Web Studio\Desktop\cprogs\2D.c - [Executing] - Dev-C++ 5.4.2

```c
#include <stdio.h>

#define M 3
#define N 4
int main (void)
{
    int A[M][N];
    int i, j;
    for(i=0; i<M;
        for(j=0; j
            A[i][j
        }
    }
    for(i=0; i<M;
        printf("Ro
        for(j=0; j
            printf
        }
        printf("\n
    }
    return 0;
}
```

C:\Users\Web Studio\Desktop\cprogs\2D.e...

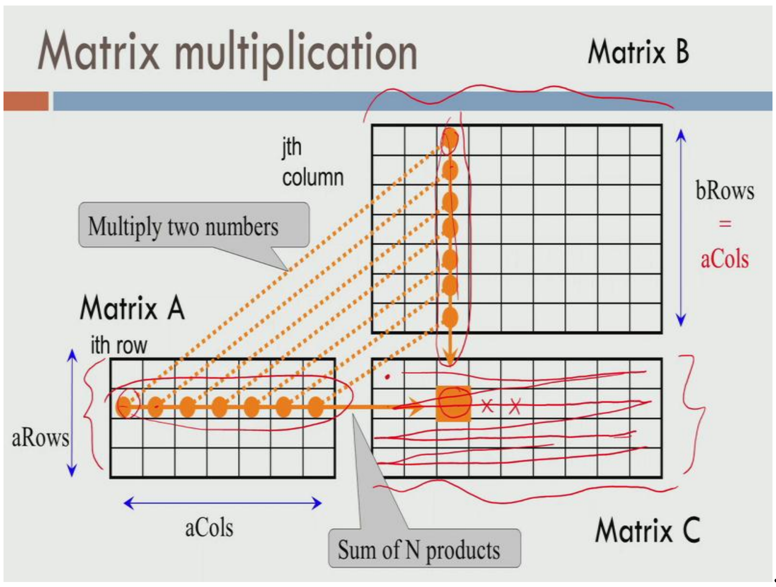
```
Row 0: 0 1 2 3
Row 1: 1 2 3 4
Row 2: 2 3 4 5

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Line: 6 Col: 2 Sel: 0 Insert Done parsing

And as I mentioned earlier, there is row 0, which has 0, 1, 2, 3; row 1, which has 1, 2, 3, 4; row 2, which has 2, 3, 4, 5 and so on. So, the basic idea is that, you can access these arrays as though you are accessing 1D array, except that, you have one dimension, which is called the major dimension; and one, which is called the minor dimension. So, the rows are the major dimensions and columns are the minor dimensions. If you have a 3D array, the plane would be the major dimension; row will be the intermediate dimension; and columns will be the minor dimension.

(Refer Slide Time: 09:28)



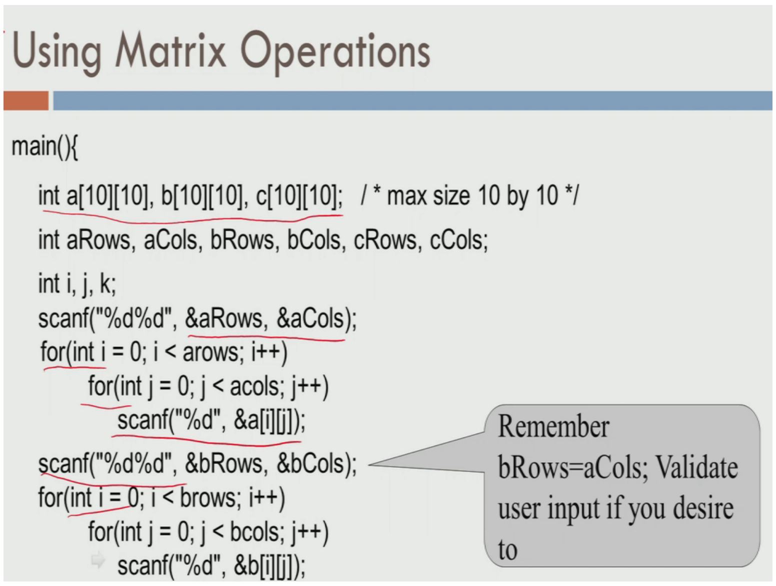
Matrix multiplication

So, what can we do with these 2D matrices? Let us see what we can do with it. So, I am going to talk about how to write a small program to do matrix multiplication. So, I am going to assume that, there is an array; there is a 2D array or a matrix called A. So, we have a matrix called A. And we have another matrix called B. Let us assume that, A has aRows number of rows and aCols number of columns. And let us assume that, B has bRows number of rows and bCols number of columns. So, if you are going to multiply A with B, we are doing A times B; the number of columns of A and the number of rows of B should match; otherwise, the matrix product would be incompatible. And how are we going to do this computation? So, you can see that, the number of rows of C are the same as the number of rows of A; and the number of columns of C is the same as the number of columns of B. So, that is the first thing that you have to do, you have to see. You can see that, this size is the same as this size; and this size is the same as this size.

So, given that, how do we now find out the actual entries of the final matrix C? So, the way you do that is as follows. See you take one element from A and you take a corresponding element from B and you multiply that. So, I took one element from A; I took a corresponding element from B. So, let us say that, I picked the i-th row from A and I picked the j-th column from B. So, i-th row will have aCols number of elements and j-th column will also have bRows equal to a columns number of elements. So, this and this dimension should match. So, the number of entries in the rows here and the number of entries in this column here should be the same.

So, what I am going to do is I am going to take one pair at a time like this; multiply these two elements and put it in the location here. Then I am going to multiply these two elements; so in the column 1 here and row 1 here. And this has to be added to the location at C and so on. So, I can proceed taking one pair of elements at a time – one element from A and another element from B. And I have to keep adding it to the elements in c of i comma j. So, essentially, the bottom line is – if you take the i-th row of A and the j-th column of B, you get the ij-th entry of C. And you do this by iterating over all the elements in the i-th row of A and j-th column of B simultaneously; you have to do this together and take one pair at a time and multiply; add it to c of i comma j. So, this is the basic logic.

(Refer Slide Time: 12:51)

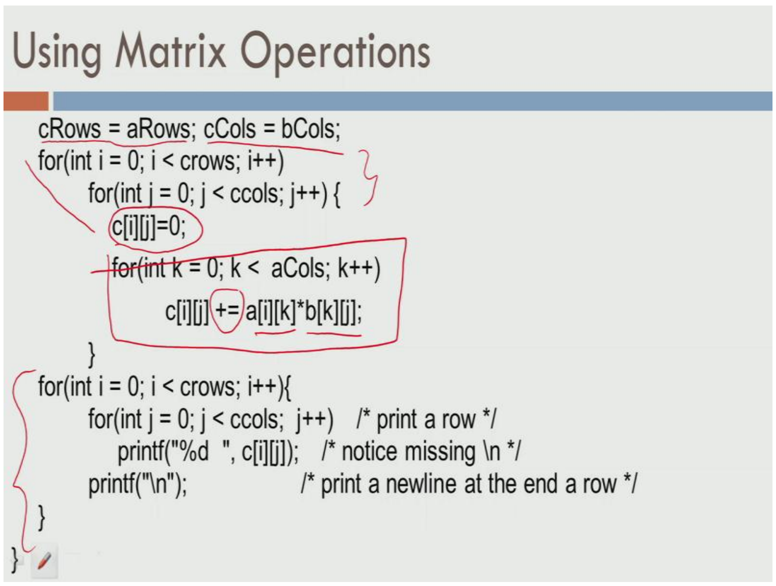


Using Matrix Operations

```
main(){
   int a[10][10], b[10][10], c[10][10];   / * max size 10 by 10 */
   int aRows, aCols, bRows, bCols, cRows, cCols;
   int i, j, k;
   scanf("%d%d", &aRows, &aCols);
   for(int i = 0; i < arows; i++)
        for(int j = 0; j < acols; j++)
           scanf("%d", &a[i][j]);
   scanf("%d%d", &bRows, &bCols);
   for(int i = 0; i < brows; i++)
        for(int j = 0; j < bcols; j++)
           scanf("%d", &b[i][j]);
```

So, let us see how to write a program for this. So, as before, we have three matrices. Let us assume that, these three matrices can accommodate up to 10 cross 10 elements. So, each of these matrices can take up to 100 elements. However, I am going to use not all the 100 elements; I may not want all the hundred elements to be filled up; I allow for a row and a set of rows and columns, that is, lesser than 10. I am going to take it from the user. So, the user is going to give aRows and aCols, which is the number of rows of A and the number of columns of A. So, clearly, A rows and A columns should both be less than or equal to 10. And what I am going do is I am going to iterate over the number of rows in A and the number of columns in A, and fill up A of i comma j. Similarly, I iterate over the number of rows of… So, I scan the number of rows from the user and columns from the user; I iterate over the set of values and end up with B filled up. And once I have A and B filled up, I am ready to do matrix multiplication.

(Refer Slide Time: 14:12)



Using Matrix Operations

```
  cRows = aRows; cCols = bCols;
  for(int i = 0; i < crows; i++)
        for(int j = 0; j < ccols; j++) {
          c[i][j]=0;
          ~~for(int k = 0; k <  aCols; k++)~~
                c[i][j] += a[i][k]*b[k][j];
        }
  for(int i = 0; i < crows; i++){
        for(int j = 0; j < ccols;  j++)    /* print a row */
           printf("%d  ", c[i][j]);   /* notice missing \n */
        printf("\n");               /* print a newline at the end a row */
  }
}
```

So, the key thing is the number of rows in C is the same as the number of rows in A; and the number of columns in C is the same as the number of columns in B. So, let us see how this program would run. So, you would need these two loops: the i-th loop and j-th loop. So, the outermost loop and the intermediate loop will iterate over all the elements c of i. j. So, this is evident from looking at this. So, if I want i comma j-th entry, I will take this row vector here and this column vector here; I will do a dot product essentially and I will put it here. Then I have to go and fill up this; then I have to go and fill up this; and so on. So, I will start from this location and I am going to start filling up elements in this order. So, this is the order in which we are going to fill up elements. So, these two loops take care of that; i equal to 0 to less than cRows, j equal to 0 j less than c columns – will iterate over all the entries of c of i, j. So, the first thing we do is initialize c of i, j to 0.

And once you have that, then we are now ready to do the actual dot product. Actual dot product is done here. I am going to look at i comma k-th entry of A and k comma j-th entry of B and I multiply that; add it to the current value of c of i, j. So, that is what the plus equal-to does. You take i comma k-th value of a, k comma j-th value of b; multiply it and add it to the current value of C. And when you do that, you see that, c of i comma j is actually accumulating the value; you have initialized to 0 and it is accumulating the value. At the end of this, c of i, j will get the final dot product of these two vectors. And you are iterating over all the values of i comma j that are permissible for C. So, at the end of it, you have all the entries in C filled up. So, this is the basic program to do matrix multiplication. And this loop is printing the entries. So, this is a fairly simple program.

So, the key thing for 2D arrays is that, you have to remember always rows followed by columns or planes followed by rows followed by columns. And you use square bracket to access the row or column; and all the numbering for the planes, for the rows, and columns start with 0. So, all the basic rules related to 1D array also applies here. So, if you the array A is of size m cross n, the largest entry you can access is A of m minus 1, n minus 1.