

CS1100 - Lecture 11

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

Two-dimensional arrays

We have a lot of situations which are naturally suitable for using higher dimensional arrays. For example, when we deal with matrices. A matrix M of order $m \times n$ has m rows and n columns. In each row of M , there are n elements. Performing matrix operations such as matrix product, various matrix transformations like exchanging two rows of a matrix etc., are easier to implement in C using two-dimensional arrays. Theoretically these problems can be solved using single dimensional arrays also, but it will be very difficult to implement.

For example, suppose we are representing an $m \times n$ matrix M using a single dimensional array a of size mn in which Row 0 elements are stored first, followed by Row 1 elements, and so on. If we use this method, the element of the matrix M in i^{th} row and j^{th} column will have index $i \times n + j$ in the single dimensional array a .

Suppose we want to exchange Row 0 and Row 2 of M . we will have to make the following exchanges:

$a[0]$ with $a[2n]$, $a[1]$ with $a[2n+1]$, $a[2]$ with $a[2n+2]$, ..., $a[n-1]$ with $a[3n-1]$.

But when we are using two-dimensional arrays, this task is very simple. This is because we can directly go to the beginning of each row easily. Moreover, as in the case of single dimensional arrays, we can directly access each element of the two dimensional array using the indices of the element.

If M is a two-dimensional array, then $M[0][0]$ represents the element in 0^{th} row and 0^{th} column and $M[0][1]$ represents element in 0^{th} row and 1^{st} column. Similarly, $M[1][0]$ represents the element in 1^{st} row and 0^{th} column. In general, the element in i^{th} row and j^{th} column can be represented by $M[i][j]$. For assigning a value 50 to an element $M[i][j]$ of the array, we can use an instruction $M[i][j]=50$.

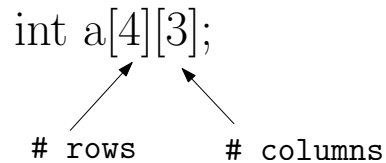
The problem of exchanging Row 0 and Row 2 of M is now easier. Exchange $M[0][0]$ with $M[2][0]$, $M[0][1]$ with $M[2][1]$, ..., $M[0][n-1]$ with $M[2][n-1]$. This can be done as follows.

```
for(i=0; i<n; i++)
{
    t=M[0][i];
    M[0][i]=M[2][i];
    M[2][i]=t;
}
```

Remember that, the solution for the same problem with a single dimensional array needed a more complicated way of calculating the indices of elements to be exchanged.

An example of declaring a two-dimensional array in C is given below.

```
int a[4][3];
```



rows # columns

A program to assign some values to elements of a 4×3 matrix and then rotate the rows of the matrix is given below. Row 0 will be replaced by Row 1, Row 1 will be replaced by Row 2, Row 2 will be replaced by Row 3 and Row 3 will be replaced by Row 0.

```
#include<stdio.h>
int main()
{
    int i, j, a[4][3], t[3];

    /* Here a can be thought of as an array with 4 elements,
       where each element of a is itself an array of 3 integers.

       a can be also understood as a 4 X 3 matrix */

    for(i=0; i<4; i++)
    {
        for(j=0; j<3; j++)
        {
            a[i][j]= i*i+j;
        }
    }
    printf("----- \n");
    printf("Matrix \n");
    printf("----- \n");
    for(i=0; i<4; i++)
    {
        for(j=0; j<3; j++)
        {
            printf("%2d ", a[i][j]);
        }
        printf("\n");
    }

    for(j=0; j<3; j++)
    {
        t[j]=a[0][j];
    }

    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
```

```

        {
            a[i][j]=a[i+1][j];
        }
    }

    for(j=0; j<3; j++)
    {
        a[3][j]=t[j];
    }

    printf("----- \n");
    printf("Updated Matrix \n");
    printf("----- \n");

    for(i=0; i<4; i++)
    {
        for(j=0; j<3; j++)
        {
            printf("%2d ", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}

```

This program makes some initialization instead of taking values from the user. After executing

```

    for(i=0; i<4; i++)
    {
        for(j=0; j<3; j++)
        {
            a[i][j]= i*i+j;
        }
    }
}

```

the elements of **a** are as in the following matrix.

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 9 & 10 & 11 \end{bmatrix}$$

The following lines are used for displaying the elements of **a** in a matrix form.

```

    printf("----- \n");
    printf("Matrix \n");
    printf("----- \n");
    for(i=0; i<4; i++)

```

```

{
    for(j=0; j<3; j++)
    {
        printf("%2d ", a[i][j]);
    }
    printf("\n");
}

```

After execution of the above lines, the following output will be displayed.

```

-----
Matrix
-----
0  1  2
1  2  3
4  5  6
9 10 11

```

The following lines are used to copy the elements of the first row of the matrix **a** into the single dimensional array **t**.

```

for(j=0; j<3; j++)
{
    t[j]=a[0][j];
}

```

After execution of the above loop, the values of the elements of the array **t** are 0, 1, 2. The following inner loop will be used to copy elements of Row **i+1** to the corresponding elements of Row **i**.

```

for(j=0; j<3; j++)
{
    a[i][j]=a[i+1][j];
}

```

Hence the following code is used to exchange Row 0 with Row 1, Row 1 with Row 2, and Row 2 with Row 3.

```

for(i=0; i<3; i++)
{
    for(j=0; j<3; j++)
    {
        a[i][j]=a[i+1][j];
    }
}

```

After executing the above part of the program, array **a** becomes

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 9 & 10 & 11 \\ 9 & 10 & 11 \end{bmatrix}$$

After the execution of the following lines, the values of elements in the array **t** are copied to the values of the corresponding elements of Row 3 of the matrix **a**.

```
for(j=0; j<3; j++)
{
    a[3][j]=t[j];
}
```

After executing the above lines, the matrix **a** is as follows.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 9 & 10 & 11 \\ 0 & 1 & 2 \end{bmatrix}$$

The lines after this are used to display the updated matrix. These lines produce the following output.

```
-----
Updated Matrix
-----
1  2  3
4  5  6
9 10 11
0  1  2
```

Memory allocation for two-dimensional arrays

int a[4][3]	2000	a[0][0]	
	2004	a[0][1]	
	2008	a[0][2]	
	200C	a[1][0]	
	2010	a[1][1]	
	2014	a[1][2]	
	2018	a[2][0]	
	.	.	
	.	.	
	202C	a[3][2]	

As in single dimensional arrays, the memory locations for storing elements of a two-dimensional arrays are also contiguous. Elements of row 0 are stored one by one and then elements of row 1 are stored one by one and so on. The above figure explains this with addresses shown in hexadecimal.

Technically, a two-dimensional array is an array of single dimensional arrays. If we have declared an array `int a[m][n]`, then `a` is an array of `m` elements, where each of the `m` elements can hold an address. Here, each of `a[0]`, `a[1]`, ..., `a[m-1]` is an array of `n` integers. Elements of the array `a[i]` are the elements of the i^{th} row of the two-dimensional array `a`. The value of the expression `a[0]` is the address of the first element of row 0 of `a` (i.e., `&a[0][0]`), the value of the expression `a[1]` is the address of the first element of row 1 (i.e., `&a[1][0]`) and so on. In general, the value of the expression `a[i]` is the address of the first element of row `i` (i.e., `&a[i][0]`).

In the example shown in the figure, `a[0]` is an array of 3 integers (`a[0][0]`, `a[0][1]`, `a[0][2]`) and the value of the expression `a[0]` is 2000, which is the address of `a[0][0]` (i.e., the value of `&a[0][0]`). `a[1]` is also an array of 3 integers and the value of the expression `a[1]` is 200C, which is the address of `a[1][0]` (i.e., the value of `&a[1][0]`).

We can access the location of each element through pointers. As in the case of single dimensional arrays, the expression `(a+i)` has value `&a[i]`. Hence, `*(a+i)` corresponds to `*(&a[i])`, which is equal to the value of `a[i]`. Thus, both `*(a+i)` and `a[i]` have value same as the address of `&a[i][0]`. Similarly, values of `(a[i]+j)` and `*(a[i]+j)` are equal to the `&a[i][0] + (j × number of locations needed to store one integer)`. This address is the same as the address of `a[i][j]`.

Since, the expression `*(a+i)+j` has value `&a[i][j]` as explained in the previous paragraph, `*(*(a+i)+j)` is the same as `*(&a[i][j])`, which is also the same as `a[i][j]`. Similarly, `*(a[i]+j)` is the same as `a[i][j]`. Thus, the array element present in the location i^{th} row and j^{th} column can be represented by either `a[i][j]`, `*(*(a+i)+j)` or `*(a[i]+j)`.

The following program demonstrates accessing elements of a two-dimensional matrix using the different methods described above.

```
#include<stdio.h>
int main()
{
    int i, j, a[4][3];

    /* Here a is an array with 4 elements,
       where each element of a is itself an array of 3 integers
       a can be understood as a 4 X 3 matrix */
    printf("Memory Allocation Details \n");
    printf("----- \n");
    for(i=0; i<4; i++)
    {
        for(j=0; j<3; j++)
        {
            printf("&a[%d][%d] is %p \n",i, j, &a[i][j]);
        }
    }

    printf("----- \n");
    printf("Row Addresses \n");
    printf("----- \n");
    for(i=0; i<4; i++)
    {
```

```

    printf("a[%d] is %p , *(a+%d) is %p\n",i, a[i], i, *(a+i));
}
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        a[i][j]= i*i+j;
    }
}
printf("----- \n");
printf("Matrix \n");
printf("----- \n");

for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        printf("%2d ", a[i][j]);
    }
    printf("\n");
}

printf("----- \n");
printf("Matrix accessed via addresses - first way \n");
printf("----- \n");
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        printf("%2d ", *(a[i]+j) );
    }
    printf("\n");
}

printf("----- \n");
printf("Matrix accessed via addresses - second way \n");
printf("----- \n");
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        printf("%2d ", (*(a+i)+j) );
    }
    printf("\n");
}
return 0;
}

```

Consider the following part of the above program.

```
printf("Memory Allocation Details \n");
printf("----- \n");
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        printf("&a[%d][%d] is %p \n",i, j, &a[i][j]);
    }
}
```

After executing the above instructions, addresses of each element of the two-dimensional array will be displayed. A sample output is shown below.

```
Memory Allocation Details
-----
&a[0][0] is 0x7ffd5cb3bda0
&a[0][1] is 0x7ffd5cb3bda4
&a[0][2] is 0x7ffd5cb3bda8
&a[1][0] is 0x7ffd5cb3bdac
&a[1][1] is 0x7ffd5cb3bdb0
&a[1][2] is 0x7ffd5cb3bdb4
&a[2][0] is 0x7ffd5cb3bdb8
&a[2][1] is 0x7ffd5cb3bdbc
&a[2][2] is 0x7ffd5cb3bdc0
&a[3][0] is 0x7ffd5cb3bdc4
&a[3][1] is 0x7ffd5cb3bdc8
&a[3][2] is 0x7ffd5cb3bdcc
```

Consider the following part of the previous program.

```
printf("-----
printf("Row Addresses \n");
printf("-----
for(i=0; i<4; i++)
{
    printf("a[%d] is %p , *(a+%d) is
}
```

After executing the above instructions, the addresses of the first elements in each row of the array will be displayed as given below.

```
-----
Row Addresses
-----
a[0] is 0x7ffd5cb3bda0 , *(a+0) is 0x7ffd5cb3bda0
a[1] is 0x7ffd5cb3bdac , *(a+1) is 0x7ffd5cb3bdac
a[2] is 0x7ffd5cb3bdb8 , *(a+2) is 0x7ffd5cb3bdb8
a[3] is 0x7ffd5cb3bdc4 , *(a+3) is 0x7ffd5cb3bdc4
```


The following lines of code assigns some values to the elements of the matrix.

```
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        a[i][j]= i*i+j;
    }
}
```

After executing the above lines of code, the matrix is as follows.

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 9 & 10 & 11 \end{bmatrix}$$

The following parts of the program shows one method of accessing array elements using address.

```
printf("Matrix accessed via addresses - first way \n");
printf("_____ \n");
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        printf("%2d ", *(a[i]+j) );
    }
    printf("\n");
}
```

After executing the above instructions, the following output will be displayed.

```
-----
Matrix accessed via addresses - first way
-----
0  1  2
1  2  3
4  5  6
9 10 11
```

The following instructions of the program shows another method of accessing array elements using address.

```
printf("----- \n");
printf("Matrix accessed via addresses - second way \n");
printf("----- \n");
for(i=0; i<4; i++)
{
    for(j=0; j<3; j++)
    {
        printf("%2d ", (*(a+i)+j) );
    }
    printf("\n");
}
```

After executing the above instructions, the following output will be displayed.

```
-----
Matrix accessed via addresses - second way
-----
0  1  2
1  2  3
4  5  6
9 10 11
```