Computer Programming & Problem Solving

CS100

Mrs Sanga G. Chaki
Department of Computer Science and Engineering
National Institute of Technology, Goa
May, 2023

Topics

1. Two Dimensional Arrays
2. Array of pointers
3. 3D arrays

2-D Arrays/Matrix

1. We have seen that an array variable can store a list of values.
2. Many applications require us to store a *table* of values.

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Student 1	75	82	90	65	76
Student 2	68	75	80	70	72
Student 3	88	74	85	76	80
Student 4	50	65	68	40	70

1. The table can be regarded as a matrix consisting of 4 rows and 5 columns.
2. C allows us to define such tables of items by using two-dimensional arrays.

Declaring 2-D Arrays

General form :

```
type  array_name [row_size][column_size];
```

Examples:

```
int marks[4][5];

float sales[12][25];
```

Accessing 2-D Array Elements

1. Similar to that for 1-D array, but use two indices.
   a) First indicates row, second indicates column.
   b) Both the indices should be expressions which evaluate to integer values.

Example:

```
int x[20][100];

x[0][0] = 36;

x[4][9] = 10;

x[19][99] = 91;
```

Initializing 2-D Arrays

```
int  stud[4][2] = {
                    { 1234, 56 },
                    { 1212, 33 },
                    { 1434, 80 },
                    { 1312, 78 }
                  } ;
```

OR

```
int stud[4][2] = { 1234, 56, 1212, 33, 1434, 80, 1312, 78 } ;
```

It is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional – when initializing a 2D array.

What happens here?

```
main( )
{
        int  stud[4][2] ;
        int  i, j ;

        for ( i = 0 ; i <= 3 ; i++ )
        {
                printf ( "\n Enter roll no. and marks" ) ;
                scanf ( "%d %d", &stud[i][0], &stud[i][1] ) ;
        }

        for ( i = 0 ; i <= 3 ; i++ )
                printf ( "\n%d %d", stud[i][0], stud[i][1] ) ;
}
```

row no. 0

row no. 1

row no. 2

row no. 3

col. no. 0	col. no. 1
1234	56
1212	33
1434	80
1312	78

Memory Map of 2D Array

1. Memory doesn't contain rows and columns.
2. In memory whether it is a one-dimensional or a two-dimensional array the array elements are stored in one continuous chain

s[0][0]	s[0][1]	s[1][0]	s[1][1]	s[2][0]	s[2][1]	s[3][0]	s[3][1]
1234	56	1212	33	1434	80	1312	78
65508	65510	65512	65514	65516	65518	65520	65522

Memory Map of 2D Array

1. Starting from a given memory location, the elements are stored row-wise in consecutive memory locations

a[0]0] a[0][1] a[0]2] a[0][3]	a[1][0] a[1][1] a[1][2] a[1][3]	a[2][0] a[2][1] a[2][2] a[2][3]
Row 0	Row 1	Row 2

Memory Map of 2D Array

- x: starting address of the array in memory
- c: number of columns
- k: number of bytes allocated per array element

- a[i][j] is allocated memory location at address x + (i * c + j) * k

s[0][0]	s[0][1]	s[1][0]	s[1][1]	s[2][0]	s[2][1]	s[3][0]	s[3][1]
1234	56	1212	33	1434	80	1312	78
65508	65510	65512	65514	65516	65518	65520	65522

Reading Elements into 2D Array

By reading them one element at a time

```
for  (i=0; i<nrow; i++)
    for  (j=0; j<ncol; j++)
        scanf  ("%f", &a[i][j]);
```

Printing Elements from 2D Array

```
for  (i=0; i<nrow; i++) {
        for  (j=0; j<ncol; j++) printf  ("%f  ", a[i][j]);
          printf("\n");
}
```

- The elements are printed with one row in each line.

Array of Pointers

1. The way there can be an array of integers or an array of floats, similarly there can be an array of pointers.
2. A collection of addresses.
3. The addresses present in the array of pointers can be addresses of isolated variables or addresses of array elements or any other addresses.
4. All rules that apply to an ordinary array apply to the array of pointers as well

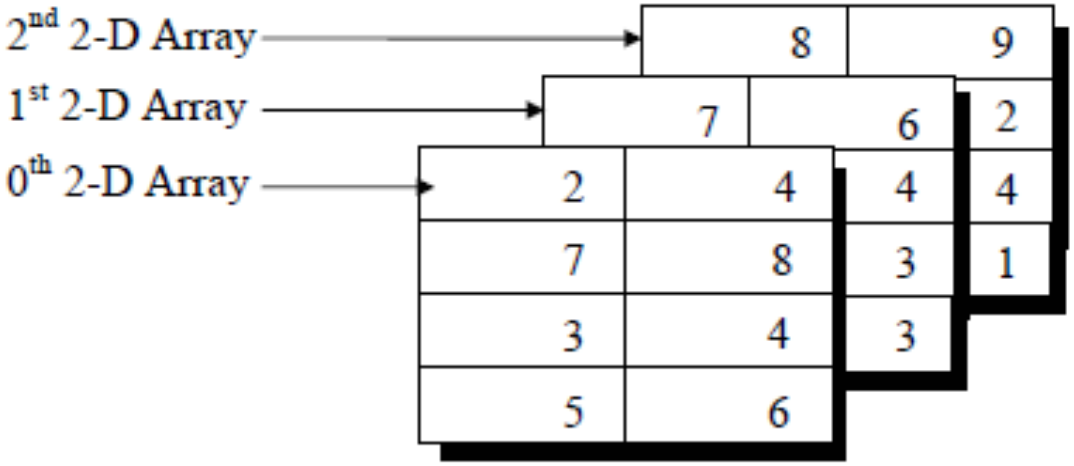
Array of Pointers

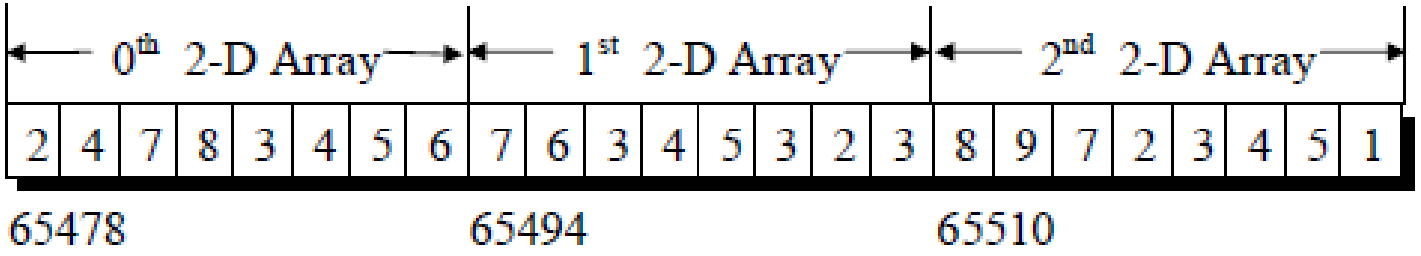
```
main( )
{
      int  *arr[4] ;  /* array of integer pointers */

       int  i = 31, j = 5, k = 19, l = 71, m ;

       arr[0] = &i ;
       arr[1] = &j ;
       arr[2] = &k ;
       arr[3] = &l ;

       for ( m = 0 ; m  <= 3 ; m++ )
             printf ( "%d ", * ( arr[m] ) ) ;
}
```

What is the output?

3-D Array

1. **An array of arrays of arrays.**





3-D Array - Initializing

```
int  arr[3][4][2] = {
                        {
                            { 2, 4 },
                            { 7, 8 },
                            { 3, 4 },
                            { 5, 6 }
                        },
                        {
                            { 7, 6 },
                            { 3, 4 },
                            { 5, 3 },
                            { 2, 3 }
                        },
                        {
                            { 8, 9 },
                            { 7, 2 },
                            { 3, 4 },
                            { 5, 1 },
                        }
};
```

1. **So, what is arr[2][3][1]?**