Two Dimensional Arrays

Rab Nawaz Jadoon

Assistant Professor

COMSATS IIT, Abbottabad

Pakistan

Department of Computer Science

DCS

COMSATS Institute of
Information Technology

Passing Array elements to function

- It is also possible for arrays to have two or more dimensions.
  - The two dimensional array is also called a matrix.

Simple Program

```
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] ) ;
}
```

Cont…

- The previous example array arrangement is as under,

	col. no. 0	col. no. 1
row no. 0	1234	56
row no. 1	1212	33
row no. 2	1434	80
row no. 3	1312	78

Initializing two dimensional arrays

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

or even this would work...

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

Initializing two dimensional arrays

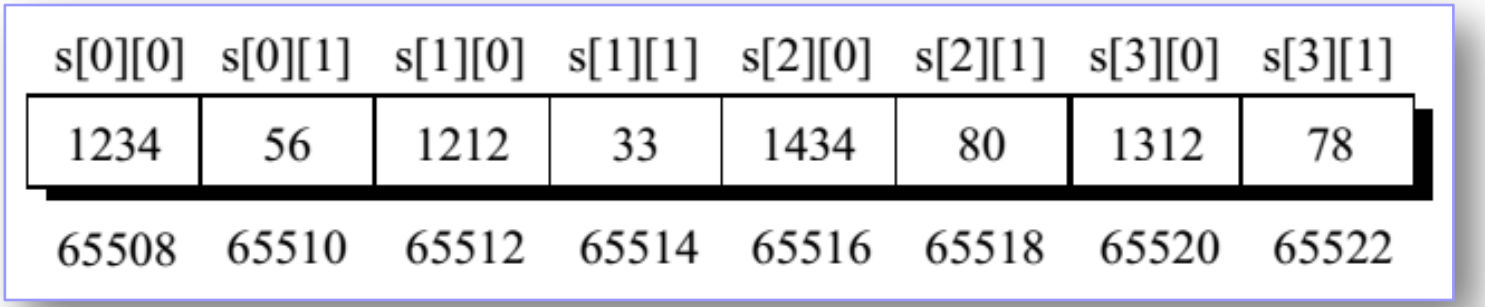
- It is important to remember that while initializing a 2-D array,
  - It is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

```
int  arr[2][3] = { 12, 34, 23, 45, 56, 45 } ;
int  arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ;
```

```
int  arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ;
int  arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;

would never work.
```

Two dimensional arrays in memory

- Memory doesn't contain rows and columns.
  - In memory whether it is a one-dimensional or a two-dimensional array the array elements are stored in one continuous chain.
  - The arrangement of array elements of a two-dimensional array in memory is shown below:



Pointers and 2-Dimensional Arrays

- Each row of a two-dimensional array can be thought of as a one-dimensional array.

```
/* Demo: 2-D array is an array of arrays */
main( )
{
      int  s[4][2] = {
                              { 1234, 56 },
                              { 1212, 33 },
                              { 1434, 80 },
                              { 1312, 78 }
                        } ;
      int  i ;

      for ( i = 0 ; i <= 3 ; i++ )
            printf ( "\nAddress of %d th 1-D array = %u", i, s[i] ) ;
}
```

```
Address of 0 th 1-D array = 65508
Address of 1 th 1-D array = 65512
Address of 2 th 1-D array = 65516
Address of 3 th 1-D array = 65520
```

Output Philosphy

- The compiler knows that s is an array containing 4 one-dimensional arrays, each containing 2 integers.
- Each one-dimensional array occupies 4 bytes (two bytes for each integer).
- These one-dimensional arrays are placed linearly (zeroth 1-D array followed by first 1-D array, etc.).

Accessing 2 dimensional array elements using pointers

```
/* Pointer notation to access 2-D array elements */
main( )
{
      int  s[4][2] = {
                              { 1234, 56 },
                              { 1212, 33 },
                              { 1434, 80 },
                              { 1312, 78 }
                        } ;
      int  i, j ;

      for ( i = 0 ; i <= 3 ; i++ )
      {
            printf ( "\n" ) ;
            for ( j = 0 ; j <= 1 ; j++ )
                  printf ( "%d ", *( *( s + i ) + j ) ) ;
      }
}
```

And here is the output...

```
1234  56
1212  33
1434  80
1312  78
```

Passing 2-D Array to a Function

```
/* Three ways of accessing a 2-D array */

main( )
{
      int  a[3][4] =  {
                           1, 2, 3, 4,
                           5, 6, 7, 8,
                           9, 0, 1, 6
                      } ;

      clrscr( ) ;
      display ( a, 3, 4 ) ;
      show ( a, 3, 4 ) ;
      print ( a, 3, 4 ) ;
}
```

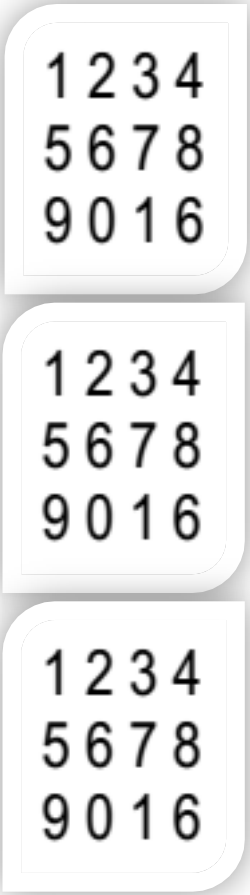
```
display ( int  *q, int  row, int  col )
{
      int  i, j ;

      for ( i = 0 ; i < row ; i++ )
      {
            for ( j = 0 ; j < col ; j++ )
                  printf ( "%d ", * ( q + i * col + j ) ) ;


            printf ( "\n" ) ;
      }
      printf ("\n" ) ;
}
```

Passing 2-D Array to a Function

```
show ( int  ( *q )[4], int  row, int  col )
{
      int  i, j ;
      int  *p ;

      for ( i = 0 ; i < row ; i++ )
      {
            p = q + i ;
            for ( j = 0 ; j < col ; j++ )
                  printf ( "%d ", * ( p + j ) ) ;

            printf ( "\n" ) ;
      }
      printf ( "\n" ) ;
}
```

```
print ( int  q[ ][4], int  row, int  col )
{
      int  i, j ;

      for ( i = 0 ; i < row ; i++ )
      {
            for ( j = 0 ; j < col ; j++ )
                  printf ( "%d ", q[i][j] ) ;
            printf ( "\n" ) ;
      }
      printf ( "\n" ) ;
}
```

Output ...



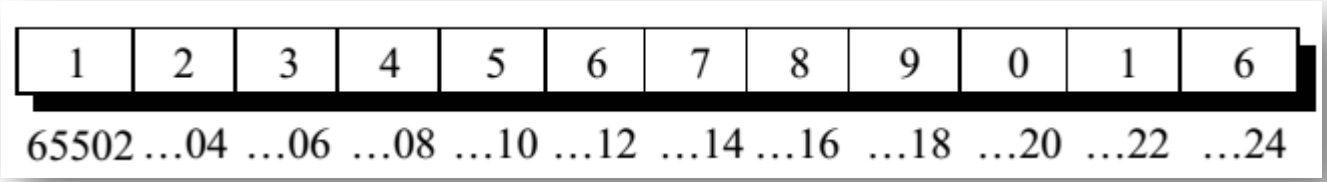
A more general formula for accessing each array element would be:

* (base address + row no. * no. of columns + column no.)

Illustration

- In the previous program the display function definition we have,
  - In the display( )function we have collected the base address of the 2-D array being passed to it in an ordinary int pointer.
  - Then through the two for loops using the expression * (q + i * col + j), we have reached the appropriate element in the array.
    - Suppose i is equal to 2 and j is equal to 3, then we wish to reach the element a[2][3]. Let us see whether the expression * (q + i* col + j) does give this element or not. Refer Figure 8.7 to understand this.

Illustration



- The expression * (q + i *col + j) becomes * (65502 + 2 * 4 + 3).
- This turns out to be *(65502 + 11).
  - Since 65502 is address of an integer,* (65502 + 11) turns out to be * (65524).
- Value at this address is 6.
- This is indeed same as a[2][3].

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] ) ) ;
}
```

Output…
31 5 19 71

Array of pointers

- An array of pointers can even contain the addresses of other arrays.

```
#include<stdio.h>
#include<conio.h>
main( )
{
 static int a[ ] = { 0, 1, 2, 3, 4 } ;
 int *p[ ] = { a, a + 1, a + 2, a + 3, a + 4 } ;
 for(int i=0;i<=4;i++)
 {
 printf ( "\n%u %u %d", (p+i), *(p+i), * ( *(p+i) ) ) ;
 }
 getche();
}
```

Output:

```
1638208  4239604   0
1638212  4239608   1
1638216  4239612   2
1638220  4239616   3
1638224  4239620   4
```


Thank You!