Array - 1D

Arrays is a simple data structure that is a collection of 1 or more values arranged in a linear fashion called as one dimensional array. Each element is accessed using an index or key, bu generally in C, we call it index. The index starts from 0 , not 1. If there are 10 elements in the array, the index starts at 0 to index 9.

In C/C++, all elements of an array are same data type.

For example,

unsigned int data1 [4] ;

means data1 is the name of the array, it contains 4 elements and the data type is unsigned int. Index starts from 0 to 3.

You can individually assign values to the cells. For instance,

data1 [0] = 9 ;

data1 [1] = 19 ;

data1 [2] = 30 ;

data1 [3]= 8 ;

char data2 [2] ;

means data2 is the name of the array, it contains 2 elements/cells and the data type is all char. We assign the values to a char array something like this.

data2 [0] = 'A' ;

data2 [1] = 66 ; // 'B'

Please note that the cells in an array are contiguous in physical memory too. That means the address of data1 [0] is very next to data1 [1] that is next to data1 [2] and data1 [3] . This is guaranteed by the system.

Unlike Java, out of bound runtime errors will crash your program without any message.

data1 [4] will give runtime error

data1[-1] will give runtime error too

data2 [4] will also give runtime error. So, care must be taken to check of range of indices. There is no such try and catch like you have in Java. It is your responsibility to make sure you always access cells within the range.

How do we Initialize Arrays

You can initialize an array using a for loop , say

```
int data [ 4 ] ;
for ( i = 0 ; i < 4 ; i++ )
    data [ i ] = 0 ;
```

or Initializing an Array in a Definition with an Initializer List

- The elements of an array can also be initialized when the array is defined by following the definition with an equals sign and braces, {}, containing a comma-separated list of array initializers.

```
int data [ 4 ] = {  3, 5, 8, 0 } ;
```

This is valid declaration making the first cell set to 3, the second cell set to 5, the third cell to 8, and the fourth cell set to 0.

You can also declare the above declation as

```
int data [  ] = {  3, 5, 8, 0 } ;  // it is okay,  it is missing number between  [  ]
```

Here the size of the array is defined after processing the number of values defined within the curly braces.

- If there are *fewer* initializers than elements in the array, the remaining elements are initialized to zero.

```
int data [ 4 ]  =  {  3, 5 } ;
```

in this case, the cell at index 0 is set to 3, cell at index at 1 is set to 5, but the rest of the cells (at index 2 and 3) are set to zero

- // initializes entire array to zeros

int data[4] = {0} ;

This *explicitly* initializes the cell at the first element to zero, but sets the remaining three elements (at index 1, 2, 3) also to zero because there are fewer initializers than there are elements in the array.

int data[4] = {2} ;

In this case, this *explicitly* initializes the cell at the first element to two, but sets the remaining three elements (at index 1, 2, 3) to zero , not 2. **VERY IMPORTANT**

Most programmers make a mistake that arrays are automatically set to zero. It is not, arrays are not automatically initialized to zero. You must at least initialize the first element to zero for the remaining elements to be automatically zeroed.

Do not Do THIS:

Here is code that yields compiler error

int data [4] = { 3, 5, 7, 2, 4, 9 } ;

why ? Here data is declared with four cells, but initialized using 6 values, which is incorrect. The correct way in this case is

int data [6] = { 3, 5, 7, 2, 4, 9 } ;

or a better way is

int data [ ] = { 3, 5, 7, 2, 4, 9 } ; // array is automatically declared with 6 cells.

Initialize in any order:

C99 version (or gcc extensions) allows to initialize the cells in any order

for instance ,

int a[4] = { [2] = 29, [0] = 15 } ;

where index 2 is set to 29, index 0 is set to 15 and the rest of the array is set to zero.

DECLARING ARRAYS USING CONSTANTS

Generally it is customary to use constant to declare an array. That is,

#define NUM_OF_CELLS 10

The above line can be used to declare an array like

int data [NUM_OF_CELLS] ;

In the above line , the compiler replaces NUM_OF_CELLS with 10 (textual substitution) , making the line

int data [10] ;

This is done at the preprocessor time.

Arrays-2D (declaration and initialization)

```
#define ROWS 3

#define COLS 4



int main ( )

{

int i, j;

int data[ROWS][COLS] ; // just definition

int data[ROWS][COLS] = { 0 } ; // Initialize all cells to zero


// In the next definition,  it initializes the first row with 3 4 1 2 values, second row with 4 2 1 1 ,  third row as 5 2 1 2 values.

int data[ROWS][COLS] = {

                    { 3, 4, 1, 2 },

                    { 4, 2, 1, 1 },

                    { 5, 2, 1, 2 }  // PAY ATTENTION,  NO SEMICOLON to the LAST ROW

            } ;  //  YOU NEED SEMICOLON HERE THOUGH


// ANOTHER WAY INITIALIZE,  here the first row is initialized to all zero,  second row to 5 6 7 9.  third row with 9 3 in the first and second column, zero in the third and fourth column

int data[ROWS][COLS] = { { 0 },

                    { 5, 6, 7, 9 },

                    { 9, 3 }   // PAY ATTENTION,  NO SEMICOLON to the LAST ROW

            } ;
```

How to print 2D array ?

You need nested for loop.

```
for ( i = 0 ; i < ROWS ; i++ )
 {
   for ( j = 0 ; j < COLS ; j++ )
          printf ( "%d \n",  data[ i ] [ j ] ) ;
    printf ( "\n") ;  // You need this to print the end of line after printing each column
}
```

Remember: data is still a pointer constant.

Main Points on Arrays

1. The name of the array is a pointer constant. In other words, the name itself is a pointer and is always points to the first element.

2. The name of the array cannot be made to point elsewhere

3. The size of the array is always : number of cells times the size of the data type of the array.

ie for array of N cells of type char, it is N times sizeof (char)

for array of N cells of type int , it is N times sizeof (int)

for array of N cells of type double, it is N times sizeof (double)

or The generic formula is :

int number_of_cells = sizeof (array) / sizeof (array[0]);

4. The first index is always 0

5. Cells of an array to contiguous and it is guaranteed.

6. You should ways initialize array by yourself.

7. You can print the address of each cell using the %p . For instance if data is the name,

```
printf ( " %p \n ",   &data[ 0 ] ) ;
```

8. Though you can declare array having any dimension, internally they are stored as 1D array. Your 2D indices would get converted into 1D index at runtime.

For instance for an array of [5] [4] , if we want to access [i] [j] cell, then the actual cell is at (4 - i) * 5 + j

This is because there is no such thing as 2D memory in our system. All memory locations are linear as 1D

9. Just like Java objects are passed as reference, arrays are passed to functions as reference. More on this when we deal with functions.

10. You cannot change the dimension/size of an array once it is declared. If you want to change the size, you are out of luck.

11. If data is the name of an array, the address of the first cell can be printed as &data[0] or data

Multidimensional Arrays

- Arrays in C can have multiple indices.
- A common use of multidimensional arrays is to represent tables of values consisting of information arranged in *rows* and *columns,* In image processing, we use 2D arrays extensively.
- To identify a particular table element, we must specify two indices: The first (by convention) identifies the element's *row* and the second (by convention) identifies the element's *column*.
- Tables or arrays that require two indices to identify a particular element are called two-dimensional arrays.

- Multidimensional arrays can have more than two indices.
- This declaration int data [3] [4] is a two-dimensional array, data.
- The array contains three rows and four columns, so it's said to be a 3-by-4 array.
- In general, an array with *m* rows and *n* columns is called an *m*-by-*n* array

- A multidimensional array can be initialized when it's defined, much like a one-dimensional array.
- For example, a two-dimensional array int data [2][2] could be defined and initialized with

  int data**[2][2] = { {1, 2}, {3, 4} } ;**

- The values are grouped by row in braces.
- The values in the first set of braces initialize row 0 and the values in the second set of braces initialize row 1.
- So, the values 1 and 2 initialize elements data[0][0] and data [0] [1], respectively, and the values 3 and 4 initialize elements data[1] [0] and data [1] [1], respectively.

- *If there are not enough initializers for a given row, the remaining elements of that row are initialized to 0.* Thus,

  int data [2][2] = { {1}, {3, 4} };

would initialize data [0][0] to 1, data [0][1] to 0, data[1][0] to 3 and data[1][1] to 4.

Consider these declarations

```
int data1 [ 3 ] [ 4 ] = {  { 8, 7, 6 , 5 },
                           {4, 3, 2, 1 } ,
                           { 0xa, 0xb, 0xc, 0xd }        } ;
```

The above declaration is simple,

- The definition of data1 provides 12 initializers in three sublists.
- The first sublist initializes *row 0* of the array to the values 8, 7, 6, 5 ; and the second sublist initializes *row 1* of the array to the values 4, 3, 2, 1 and the third initialized the row 2 to 0xa, 0xb, 0xc, 0xd
- There is no comma for the last row

```
int data2 [ 3 ] [ 4 ] = {   8, 7, 6 , 5 ,
                            4, 3, 2, 1
                             } ;
```

In the above declaration,

- If the braces around each sublist are removed from the data2 initializer list, the compiler initializes the elements of the first row followed by the elements of the second row and so on.
- The definition of data2 provides eight initializers.
- The initializers are assigned to the first row, then the second row.
- Any elements that do not have an explicit initializer are initialized to zero automatically, so data2 [2] [0] and other cells in the third row are initialized to 0.

Now consider this

```
int data3 [ 3 ] [ 2 ] = { { 2, 1}, { 4 }  } ;
```

The definition of data3 provides three initializers in two sublists

The first row is initialized to 2 and 1. The second row, first cell is set to 4, but the second cell is set to 0. So, is all cells in the third row, set to 0.

As I mentioned in the class, it doesn't matter if you declare arrays as multidimensional arrays, they are still stored in one linear array. The compiler lets us access them as 1D too. Thus,

- The first index of a multidimensional array is not required either, but all subsequent indices are required.

```
int data [ ] [ 3 ] = {  {  4, 3, 1 }  , { 7, 5, 6}  } ;
```

- The compiler uses the rest of the indices to determine the locations in memory of elements in multidimensional arrays.
- All array elements are stored consecutively in memory regardless of the number of indices.
- In a two-dimensional array, the first row is stored in memory followed by the second row.
- Providing the index values in a parameter declaration enables the compiler to tell the function how to locate an element in the array.

Can you tell me how many rows are there in this next definition ?

```
int data  [  ] [ 3 ] = {  {  4, 3, 1 }  , { 7, 5, 6},  { 0 },  { 0 }  } ;
```