UNIT-1 (ARRAY) (RAM GOPAL GUPTA- http://ramgopalgupta.com/)

DEFINITION, DECLARATION AND INITIALIZATION

Basic Concept

An array is a group (or collection) of homogenous data/ same data types arranged in sequential format.

For example: an int array holds the elements of int types while a char array holds the elements of char types.

Need of array

Consider a scenario where you need to find out the sum of 100 integer numbers entered by user. In C, you have two ways to do this:

1) Define 100 variables with int data type and then perform 100 scanf() operations to store the entered values in the variables and then at last calculate the sum of them.

OR

2) Have a single integer array to store all the values, loop the array to store all the entered values in array and later calculate the sum.

Which one the best in above solutions: *Obviously solution No: 2*

How to declare Array in C

int num[15]; /* An integer array of 15 elements */

char ch[10]; /* An array of characters for 10 elements */

How to access element of an array in C (very important)

Suppose there is an array

int arr[20];

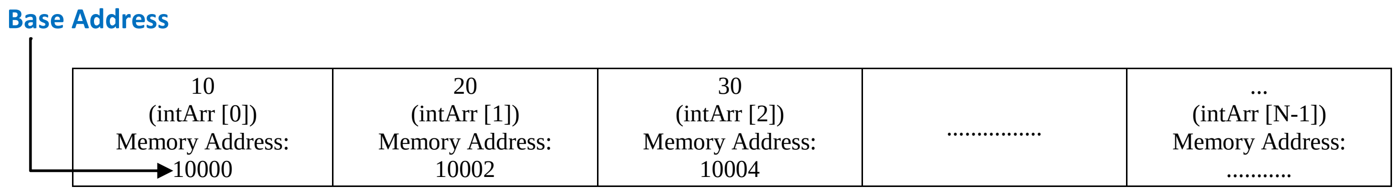
You can use array subscript (or index) to access any element stored in array. Subscript starts with 0, which means

arr[0] represents the 1 element in the array arr
arr[1] represents the 2 element in the array arr
arr[2] represents the 3 element in the array arr
.
.
.
.
arr[19] represent the 20th (Last) element in the array arr.
therefore we can say array index start from [0] and end at [n-1]; if the size of array is n.

Array Memory allocation and representation

Below diagram shows how memory is allocated to an integer array of N elements. Its base address – address of its first element is 10000. Since it is an integer array, each of its element will occupy 2 bytes of space. Hence first element occupies memory from 10000 to 10001. Second element of the array occupies immediate next memory address in the memory, i.e.; 10002 which requires another 2 bytes of space. Hence it occupies from 10002 to 10003. In this way all the N elements of the array occupies the memory space.

int intArr [N];



10 (intArr [0]) Memory Address: 10000	20 (intArr [1]) Memory Address: 10002	30 (intArr [2]) Memory Address: 10004	....	... (intArr [N-1]) Memory Address: ....

If the array is a character array, then its elements will occupy 1 byte of memory each. If it is a float array then its elements will occupy 8 bytes of memory each. But this is not the total size or memory allocated for the array. They are the sizes of individual elements in the array. If we need to know the total size of the array, then we need to multiply the number of elements with the size of individual element.

i.e.; **Total memory allocated to an Array = Number of elements * size of one element**

Total memory allocated to an Integer Array of N elements = Number of elements * size of one element
= N * 2 bytes
= 10 * 2 bytes = **20 Bytes**, where N = 10 // int num [10]
= 500 * 2 bytes = **1000 Bytes**, where N = 500 // int num [500]

Total memory allocated to an character Array of N elements= Number of elements * size of one element
= N * 1 Byte
= 10 * 1 Byte = **10 Bytes**, where N = 10 // char ch [10]
= 500 * 1 Byte = **500 Bytes**, where N=500 // char ch [500]
This is how memory is allocated for the single dimensional array.

Ways to initialize/ assign a value to an array

Initialize an array

int arr[5] = {1, 2, 3, 4 ,5};

OR

int arr[] = {1, 2, 3, 4, 5};

(both are same)

Assign a value to an array

```
int arr[10]; // array declaration

arr[0]=15;  // assign a value to [0] index of array

arr[2]=10; // assign a value to [1] index of array
```

to get a number from user and store in 0 index [0] of array use this code

```
scanf("%d",arr[0]);
```

<u>Display the array elements</u>

```
main(){

	int num[]={20,34,56,90,10};

	printf("%d\n",num[0]); // way to display value from index [0] of array num
	printf("%d\n",num[1]); // way to display value from index [1] of array num
	printf("%d\n",num[2]); // way to display value from index [2] of array num
	printf("%d\n",num[3]); // way to display value from index [3] of array num
	printf("%d\n",num[4]); // way to display value from index [4] of array num

}
```

OR

```
main(){

	int i=0;

	int num[]={20,34,56,90,10};

	for(i=0; i<5; i++){

	printf("%d\n", num[i]); // way to display value from index [i] of array num

	}

}
```

<u>Array types:</u>

There are two types of array single dimensional array and multi-dimensional array.

Here we will cover single and two-dimensional array.

Single dimensional are covered in above write-up.

Now the turn of Two-dimensional array:

An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns.

Declare the 2D array is given below.

data_type array_name[rows][columns];

int num[5] [3];

Initialization of 2D array

int num[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

Assign a value to an array

num [row] [col] = value;

num[0] [0] = 20;

num[0] [1] =13;

<u>Display the array elements</u>

Syntax: printf("%d", num[row] [column]);

Program example:

```
main(){

int i=0, j=0;

int num[4] [3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

//traversing 2D array

for(i=0;i<4;i++){

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

   printf("num[%d] [%d] = %d \n", i, j, num[i] [j]);

 }//end of j

}//end of i

}
```

<u>2D-Array Memory allocation and representation</u>

int intArr [3] [3];

Row Major Order

Let us consider a two dimensional array to explain how row major order way of storing elements works. In the case of 2D array, its elements are considered as rows and columns of a table.

When we represent an array as intArr [i] [j], the first index of it represents the row elements and the next index represents the column elements of each row.

1. When we store the array elements in row major order, first we will store the elements of first row followed by second row and so on.
2. Hence in the memory we can find the elements of first row followed by second row and so on.
3. In memory there will not be any separation between the rows.
4. We have to code in such a way that we have to count the number of elements in each row depending on its column index.
5. But in memory all the rows and their columns will be contiguous.
6. Below diagram will illustrate the same for a **2D array of size 3 X 3 i.e.; 3 rows and 3 columns. int intArr [3] [3];**
7. Array indexes always start from 0.
8. Hence the first element of the 2D array is at intArr[0][0]. This is the first row-first column element.
9. Since it is an integer array and suppose it occupies 2 bytes of space.
10. Next memory space is occupied by the second element of the first row, i.e.; intArr [0][1] – first row-second column element. This continues till all the first row elements are occupied in the memory.
11. Next it picks the second row elements and is placed in the same way as first row. This goes on till all the elements of the array are occupies the memory like below. This is how it is placed in the memory.
12. But seeing the memory address or the value stored in the memory we cannot predict which is the first row or second row or so.

Base Address

10 (Arr [0] [0]) Memory Address: 10000	20 (Arr [0] [1]) Memory Address: 10002	30 (Arr [0] [2]) Memory Address: 10004
40 (Arr [1] [0]) Memory Address: 10006	50 (Arr [1] [1]) Memory Address: 10008	60 (Arr [1] [2]) Memory Address: 10010
70 (Arr [2] [0]) Memory Address: 10012	80 (Arr [2] [1]) Memory Address: 10014	90 (Arr [2] [2]) Memory Address: 10016

Total size/ memory occupied by 2D array is calculated as

Total memory allocated to 2D Array = Number of elements * size of one element
= Number of Rows * Number of Columns * Size of one element

Total memory allocated to an Integer Array of size M x N = Number of elements * size of one element
assume an holds
=M Rows* N Columns * 2 Bytes
= 10*10 * 2 bytes = **200 Bytes**, where M =N = 10 // int num [10] [10]
= 500*5 *2 bytes= **5000 Bytes**, where M=500 and N= 5 // int num [500] [5]

Total memory allocated to an character Array of N elements= Number of elements * size of one element
= M Rows* N Columns * 1 Byte
= 10*10 * 1 Byte = **100 Bytes**, where N = 10 // char ch [10 [10]
= 500*5 * 1 Byte = **2500 Bytes**, where M=500 and N= 5 // char ch [500] [5]

Column Major Order

This is the opposite method of row major order of storing the elements in the memory. In this method all the first column elements are stored first, followed by second column elements and so on.

Base Address

10 (Arr [0] [0]) Memory Address: 10000	20 (Arr [0] [1]) Memory Address: 10006	30 (Arr [0] [2]) Memory Address: 10012
40 (Arr [1] [0]) Memory Address: 10002	50 (Arr [1] [1]) Memory Address: 10008	60 (Arr [1] [2]) Memory Address: 10014
70 (Arr [2] [0]) Memory Address: 10004	80 (Arr [2] [1]) Memory Address: 10010	90 (Arr [2] [2]) Memory Address: 10016

Total size/ memory occupied by 2D array is calculated as in the same way as above.

Total memory allocated to 2D Array = Number of elements * size of one element
= Number of Rows * Number of Columns * Size of one element

Total memory allocated to an Integer Array of size M x N = Number of elements * size of one element
=M Rows* N Columns * 2 Bytes
= 10*10 * 2 bytes = **200 Bytes**, where M =N = 10 // int num [10] [10]
= 500*5 *2 bytes= **5000 Bytes**, where M=500 and N= 5 // int num [500] [5]

Total memory allocated to an character Array of N elements= Number of elements * size of one element
= M Rows* N Columns * 1 Byte

= 10*10 * 1 Byte = 100 Bytes, where M = N = 10   // char ch [10] [10]
= 500*5 * 1 Byte = 2500 Bytes, where M=500 and N= 5   // char ch [500] [5]

FORMULA TO CALCULATE THE MEMORY ADDRESS IN 2D ARRAY:

Row major address: B + S * (N * (i-1) + j-1)

Column major address: B + S * (M * (j-1) + i-1)

- Here is an array of size M x N
- B is the base address of 2D array
- S is the size of each element in the array
- i is the row index
- j is the col index

Memory Address Representation Using Row Major

Base Address

10 (Arr [0] [0]) Memory Address: 10000	20 (Arr [0] [1]) Memory Address: 10002	30 (Arr [0] [2]) Memory Address: 10004
40 (Arr [1] [0]) Memory Address: 10006	50 (Arr [1] [1]) Memory Address: 10008	60 **(Arr [1] [2]) Memory Address: 10010**
70 (Arr [2] [0]) Memory Address: 10012	80 (Arr [2] [1]) Memory Address: 10014	90 (Arr [2] [2]) Memory Address: 10016

Memory Address Representation Using Column Major

Base Address

10 (Arr [0] [0]) Memory Address: 10000	20 (Arr [0] [1]) Memory Address: 10006	30 (Arr [0] [2]) Memory Address: 10012
40 (Arr [1] [0]) Memory Address: 10002	50 (Arr [1] [1]) Memory Address: 10008	60 **(Arr [1] [2]) Memory Address: 10014**
70 (Arr [2] [0]) Memory Address: 10004	80 (Arr [2] [1]) Memory Address: 10010	90 (Arr [2] [2]) Memory Address: 10016

Suppose an array: **int Arr [3] [3];**
now calculate the memory address of **Arr [1] [2]** in row major and col major using above formula:

Row major address: B + S * (N * (i - 1) + j - 1)
in our case we want get the address of Arr [1] [2] that means 2 row and 3 col
B = 10000, S = 2 bytes (size of int data type)
N = 3 (no. Of cols in Arr), i= 2 (row number), j = 3 (col number)

Memory address of Arr [1] [2] (in Row major order)
= 10000 + 2 * (3 * (2 - 1) + 3 - 1)
= 10000 + 2 * (3 * 1 + 3 - 1)
= 10000 + 2 * (3 + 3 – 1)
= 10000 + 2 * (3 + 2)
= 10000 + 2 * 5
= 10000 + 10
= 10010

Column major address: B + S * (M * (j - 1) + i - 1)

in our case we want get the address of Arr [1] [2] that means 2 row and 3 col

B = 10000, S = 2 bytes (size of int data type)

M = 3 (no. of rows in Arr), i= 2 (row number), j = 3 (col number)

Memory address of Arr [1] [2] (in Col major order)
= 10000 + 2 * (3 * (3 - 1) + 2 - 1)
= 10000 + 2 * (3 * 2 + 2 - 1)
= 10000 + 2 * (6 + 2 – 1)
= 10000 + 2 * (6 + 1)
= 10000 + 2 * 7
= 10000 + 14
= 10014