Lab 09: Arrays

*Exploring what are arrays, Array declaration, initialization, accessing array elements and memory representation of arrays.*

# What are Arrays

Array is collection of similar elements. These similar elements could be all ints, or all floats or all chars, etc. usually the array of characters are called ‘string’, whereas an array of ints or floats are called simply an array.

In C programming, one of the frequently arising problem is to handle similar types of data. For example: If the user want to store marks of 100 students. This can be done by creating 100 variable individually but, this process is rather tedious and impracticable. These type of problem can be handled in C programming using arrays.

An array is a sequence of data item of homogeneous value (same type). Arrays are of two types:

* + 1. One-dimensional arrays
    2. Multidimensional arrays

For now we are going to explore the one-dimensional arrays.

# 1.1 What is the need of an array

Consider a scenario wherein you have to store 100 integer numbers, entered by user, in order to find out the average of them. To program this scenario you have two ways –

1. Define 100 variable of integer type and at last perform the average operation.
2. Have a single integer array to store all the values. Which solution is better as per you?

Obviously the second solution, it is convenient to store same data types in one single variable and later access them using array index

# Array declaration

To begin with like other variable an array needs to be declared so that compiler will know what kind of an array and how large an array we want.

data\_type array\_name[array\_size];

**For example**:

int age[5];

Here,

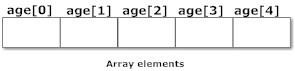
* int specifies the type of variable
* age specifies name of variable
* 5 tells how many elements of type int
* Square Brackets [ ] tells compiler that we are dealing with an array.

# Accessing elements of an array

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user as a variable according to the need of program. An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

### Example:

#### int age[5];

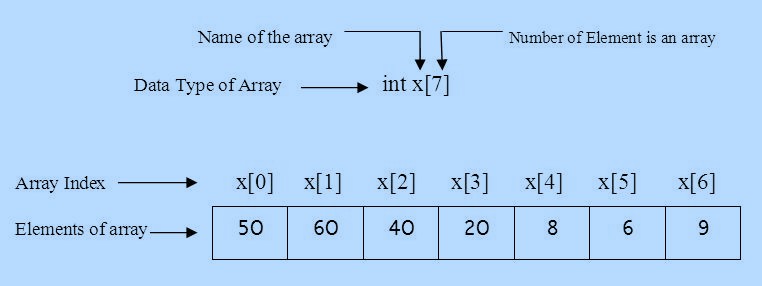


Note that, the first element is numbered 0 and so on.

Here, the size of array *age* is 5 times the size of int because there are 5 elements.

Suppose, the starting address of age[0] is 2120 and the size of int be 4 bytes. Then, the next address (address of a[1]) will be 2124, address of a[2] will be 2128 and so on.

Array elements can be referred with subscript, the number in the bracket following the array name. This number specifies the element’s position in the array. All the array elements are numbered, starting with 0, thus marks[2] is not second element of the array, but the third.



### Example:

double salary = balance[9];

The above statement will take the 10th element from the array and assign the value to salary variable. The following example shows how to use all the three above mentioned related to declaration, assignment, and accessing arrays:

#include <stdio.h> int main () {

int n[ 10 ]; /\* n is an array of 10 integers \*/

int i,j;

|  |  |  |
| --- | --- | --- |
| /\* initialize elements of array n to 0 \*/ for ( i = 0; i < 10; i++ ) {  n[ i ] = i + 100; /\* set element at location i to i + 100 \*/  }  /\* output each array element's value \*/ for (j = 0; j < 10; j++ ) {  printf("Element[%d] = %d\n", j, n[j] );  }  getch(); | | |
| } |  |  |
| **Output:** |  |  |
| Element[0] | = | 100 |
| Element[1] | = | 101 |
| Element[2] | = | 102 |
| Element[3] | = | 103 |
| Element[4] | = | 104 |
| Element[5] | = | 105 |
| Element[6] | = | 106 |
| Element[7] | = | 107 |
| Element[8] | = | 108 |
| Element[9] | = | 109 |

# Initialization of one-dimensional array

Arrays can be initialized at declaration time like this:

## int age[5]={2,4,34,3,4};

Or like below, but in this case, the compiler determines the size of array by calculating the number of elements of an array.

## int age[]={2,4,34,3,4};

Uninitialized array always contain garbage values.

# Memory Representation of Array

## int age[5]={2,4,34,3,4};

*Array Indexes*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **age[0]** | **age[1]** | **age[2]** | **age[3]** | **age[4]** |
| 2 | 4 | 34 | 3 | 4 |
| 88820 | 88824 | 88828 | 88832 | 88836 |

*Elements of Array*

*Memory Addresses*

All array elements occupy contagious space in the memory. There is a difference of 4 in the addresses of respective neighbours. This is because this array of integer type and an integer holds 4 bytes of memory.

### Example:

/\* C program to find the sum marks of n students using arrays \*/ #include <stdio.h>

int main(){

int marks[10],i,n,sum=0; printf("Enter number of students: "); scanf("%d",&n);

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

printf("Enter marks of student%d: ",i+1); scanf("%d",&marks[i]);

sum+=marks[i];

}

printf("Sum= %d",sum); getch();

}

**Output**

Enter number of students: 3 Enter marks of student1: 12 Enter marks of student2: 31 Enter marks of student3: 2

sum=45

### Task 1:

Write a program that will insert 10 integer elements in an array and print them in new lines.

### Task 2:

Write a program that will ask user to input index number from where to delete a number from array.

### Task 3:

Write a program to find out the largest and the smallest number in an array.

### Task 4:

Take 10 numbers input from user in an array, calculate the sum and average of those 10 numbers.

### Task 5:

Write a program to create two character arrays of same length and copy the content of one array into another in reverse order.