|  |  |  |
| --- | --- | --- |
| **CL1002 INTRODUCTION TO COMPUTING** | **LAB 08**  **ARRAYS IN C** | |
|  | |  |

**ARRAY**

An array is a collection of data items of the same type.

**SIGNIFICANCE OF ARRAY**

Programming problems can be solved efficiently by grouping the data items together in main memory than allocating an individual memory cell for each variable.

For Example: A program that processes exam scores for a class, would be easier to write if all the scores were stored in one area of memory and were able to be accessed as a group. C allows a programmer to group such related data items together into a single composite data structure called array.

**ONE-DIMENSIONAL ARRAYS**

In one-dimensional array, the components are arranged in the form of a list.

**SYNTAX:**

element-type aname [ size ]; /\* uninitialized \*/

element-type aname [ size ] = { initialization list }; /\* initialized \*/

**INTERPRETATION:**

* The general uninitialized array declaration allocates storage space for array aname consisting of size memory cells.
* Each memory cell can store one data item whose data type is specified by element-type (i.e., double, int , or char ).
* The individual array elements are referenced by the subscripted variables aname [0] , aname [1] , . . , aname [ size −1] .
* A constant expression of type int is used to specify an array’s size . In the initialized array declaration shown, the size shown in brackets is optional since the array’s size can also be indicated by the length of the initialization list .
* The initialization list consists of constant expressions of the appropriate element-type separated by commas.
* Element 0 of the array being initialized is set to the first entry in the initialization list , element 1 to the second, and so forth.

**MEMORY REPRESENTATION**

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

double x[5] = { 5.0, 2.0, 3.0, 1.0, -4.5};

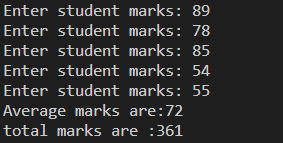
Array x

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x[0] | x[1] | x[2] | x[3] | x[4] |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x[0] | x[1] | x[2] | x[3] | x[4] |
| 5.0 | 2.0 | 3.0 | 1.0 | -4.5 |

**EXAMPLE (1D ARRAY)**

**/\* 1D Array 1st Example \*/**

**#include<stdio.h>**

**int main(){**

**int avg, sum = 0;**

**int i;**

**int marks[5]; // array declaration**

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

**printf("Enter student marks: ");**

**scanf("%d", &marks[i]); /\* stores the data in array\*/**

**}**

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

**sum = sum + marks[i]; /\* reading data from array \*/**

**avg = sum/5;**

**printf("Average marks are:%d\n", avg);**

**printf("total marks are :%d\n", sum);**

**return 0;**

**}**

**Example 02:**

#define size 5

#include <stdio.h>

int main(void)

{

int i;

/\*

int arrOfNumbers[5];

arrOfNumbers[0] = 10;

arrOfNumbers[1] = 20;

arrOfNumbers[2] = 30;

 arrOfNumbers[3] = 40;

arrOfNumbers[4] = 50;\*/

// 2nd alternattive

//int arrOfNumbers[5] = {10,20,30,40,50};

// 3rd alternative declaration

//int arrOfNumbers[] = {10,20,30,40,50};

// 4th alternative using macro

int arrOfNumbers[size] = {10,20,30,40,50};

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

{

/\* The braces are not necessary; we use them to make the code

clearer. \*/

if(arrOfNumbers[i] > 20)

printf("%d\n", arrOfNumbers[i]);

}

return 0;}

**Example 03**

**C does not check for index out of bound, it can be done by programmer itself.**

#include <stdio.h>

int main(void)

{

int i, j = 30, arrOfNumbers[3];

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

arrOfNumbers[i] = 100;

printf("%d\n", j);

return 0;

} /\* The program does not throw any error. It will print 30 once.

**Example 04**

#include <stdio.h>

int main(void)

{

int i, arrOfNumbers[10] = {0};

for(i = 0; i < 10; i++){

arrOfNumbers[++i] = 20;

// arrOfNumbers[i] = 20;

printf("The elements of array are:\t%d\n",arrOfNumbers[i]);

}

return 0;

}

**TWO-DIMENSIONAL ARRAYS**

A two dimensional array is a collection of a fixed number of components arranged in rows and columns (that is, in two dimensions), wherein all components are of the same type.

Two-dimensional arrays are used to represent tables of data, matrices, and other two-dimensional objects.

**SYNTAX:**

element-type aname [ size1 ] [ size2 ]; /\* uninitialized \*/

**INTERPRETETION**

* Allocates storage for a two-dimensional array ( aname ) with size1 rows and size2 columns.
* This array has size1\*size2 elements, each of which must be referenced by specifying a row subscript ( 0 , 1 ,… size1-1 ) and a column subscript ( 0 , 1 ,…size2-1 ).
* Each array element contains a character value.

**MEMORY REPRESENTATION**

char x[ 3 ][ 3 ] = {{‘X’, ‘0’, ‘X’}, {‘0’, ‘X’, ‘0’}, {‘0’, ‘X’, ‘X’}};

Array x

Column

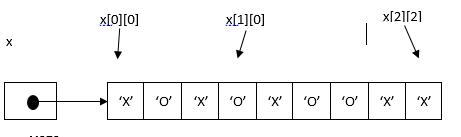
Row

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | x[0][0] | x[0][1] | x[0][2] |
| 1 | x[1][0] | x[1][1] | x[1][2] |
| 2 | x[2][0] | x[2][1] | x[2][2] |

Column

|  |  |  |  |
| --- | --- | --- | --- |
| Row | 0 | 1 | 2 |
| 0 | X | O | X |
| 1 | O | X | O  x[ 1 ][ 2 ] |
| 2 | O | X | X |

Because memory is addressed linearly, a better representation is like:



**USES**

* Storing a table of data (not the only way).
* Any kind of matrix processing, as a 2D array really is a matrix.

Example 01:

#include<stdio.h>

int main(){

/\* 2D array declaration\*/

int array[3][3];

/\*Counter variables for the loop\*/

int i, j;

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

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

printf("Enter value for array[%d][%d]:", i, j);

scanf("%d", &array[i][j]);

}

}

//Displaying array elements

printf("Two Dimensional array elements:\n");

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

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

printf("%d ", array[i][j]);

if(j==2){

printf("\n");

}

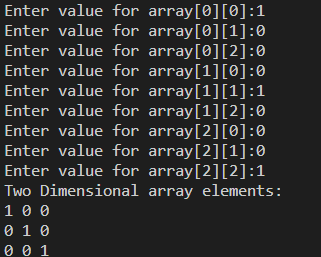
}

}

return 0;

}

Output:



**MULTIDIMENSIONAL ARRAYS**

Multidimensional array is a collection of a fixed number of elements (called components) arranged in n dimensions (n>=1).

**SYNTAX:**

element-type aname [ size 1 ] [ size 2 ] … [ size n ]; /\* storage allocation \*/

**INTERPRETATION:**

* Allocates storage space for an array aname consisting of size 1 × size 2 × … × size n memory cells.
* Each memory cell can store one data item whose data type is specified by element-type . The individual array elements are referenced by the subscripted variables aname [0][0] … [0] through aname [ size 1 −1][ size 2 −1] …[ size n −1] .
* An integer constant expression is used to specify each size i .

**USES**

With input data on temperatures referenced by day, city, county, and state, day would be the first dimension, city would be the second dimension, county would be the third dimension, and state would be the fourth dimension of the array. In any case, any temperature could be found as long as the day, the city, the county, and the state are known. A multidimensional array allows the programmer to use one array for all the data.

Example:

#include <stdio.h>

**int** main(**void**)

{

    // initializing the 3-dimensional array

**int** x[2][3][2] = { { { 0, 1 }, { 2, 3 }, { 4, 5 } },

                       { { 6, 7 }, { 8, 9 }, { 10, 11 } } };

    // output each element's value

**for** (**int** i = 0; i < 2; ++i) {

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

**for** (**int** k = 0; k < 2; ++k) {

**printf**("Element at x[%i][%i][%i] = %d\n", i, j, k, x[i][j][k]);

            }

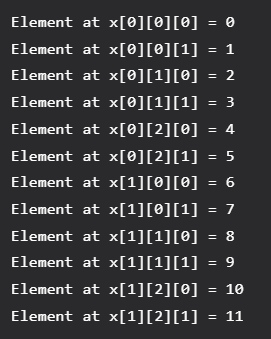
        }

    }

**return** (0);

}

Output:



**Exercises**

1. Write a C Program that takes an user input array and prints the sum of its elements.

Input: {1,2,3,4,5,6,7,8,9}

Output: 45

1. Write a program in C to read n number of values in an array and display it in reverse order

Input: {1,2,3,4,5,6,7,8,9}

Output: 9 8 7 6 5 4 3 2 1

1. Write a C Program to find the minimum and maximum number in an array.

Input: {4,1,6,8,10,21,8,9,2,6}  
Output:

Minimum Number = 1

Maximum Number = 21

1. Given an array arr[] of size N which contains elements from 0 to N-1, you need to find all the elements occurring more than once in the given array.

Input:

Array Size =5

Element 1=2

Element 2=3

Element 3=1

Element 4=2

Element 5=3

Output:  
Number 2 and 3 in array occur more than once.

1. This C program will read a matrix of nxn rows and columns and print its transpose matrix.

Input:

Enter number of Rows :2

Enter number of Cols :3

Enter matrix elements :

Enter element [1,1] : 1

Enter element [1,2] : 2

Enter element [1,3] : 3

Enter element [2,1] : 4

Enter element [2,2] : 5

Enter element [2,3] : 6

Output:

Transpose Matrix is :

1 4

2 5

3 6

1. Write a C Program that takes two nxn matrices and make a third matrix, which will contain the multiplication of both input matrices.

Input:

Enter number of Rows of matrix a: 3

Enter number of Cols of matrix a: 3

Enter elements of matrix a:

Enter element [1,1] : 1

Enter element [1,2] : 2

Enter element [1,3] : 3

Enter element [2,1] : 4

Enter element [2,2] : 5

Enter element [2,3] : 6

Enter element [3,1] : 7

Enter element [3,2] : 8

Enter element [3,3] : 9

Enter number of Rows of matrix b: 3

Enter number of Cols of matrix b: 3

Enter elements of matrix b:

Enter element [1,1] : 1

Enter element [1,2] : 1

Enter element [1,3] : 1

Enter element [2,1] : 2

Enter element [2,2] : 2

Enter element [2,3] : 2

Enter element [3,1] : 3

Enter element [3,2] : 3

Enter element [3,3] : 3

Output:

Matrix after multiplying elements (result matrix):

14 14 14

32 32 32

50 50 50

1. Given an unsorted array A of size N that contains only non-negative integers, find a continuous sub-array which adds to a given number S. In case of multiple subarrays, return the subarray which comes first on moving from left to right.

Input:

Enter Number =12

Enter Length of array = 5

Element 1: 1

Element 2: 2

Element 3: 3

Element 4: 7

Element 5: 5

Output:

The Elements from Position 2 to 4 when summed results in the output of 12.

1. Write a C Program to find the trace of a matrix. Trace of a nxn matrix is sum of diagonal elements. Give a square matrix we have to find the trace of the matrix

Input:

Matrix:

9 8 7

5 4 6

1 2 3

Output:

Trace of matrix is: 16

1. Given a matrix Write a C program which takes row numbers from the user and interchange them at the specified rows provided by the user.

Input:

Matrix before row exchange:

1 2 3

4 5 6

7 8 9

Enter two row numbers to be exchanged:1 3

Output:

Matrix after row exchange:

7 8 9

4 5 6

1 2 3