CL1002
INTRODUCTION TO COMPUTING

LAB 08
Nested Iterations and 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

```c
/* 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;

}
```
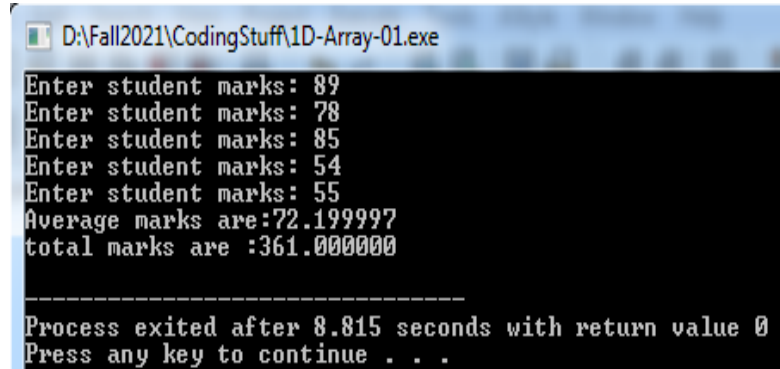
```
D:\Fall2021\CodingStuff\1D-Array-01.exe

Enter student marks: 89
Enter student marks: 78
Enter student marks: 85
Enter student marks: 54
Enter student marks: 55
Average marks are:72.199997
total marks are :361.000000

----------------------------------------
Process exited after 8.815 seconds with return value 0
Press any key to continue . . .
```

Example 02:

```c
#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.
```c
#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

```c
#include <stdio.h>
int main(void)
{
    int i, arrOfNumbers[10] = {0};
    for(i = 0; i < 10; i++){
            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 [size$_1$] [size$_2$]; /* 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
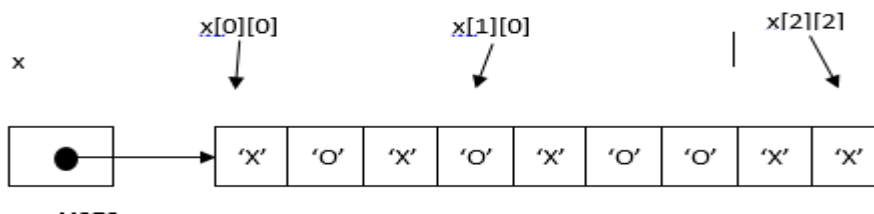char x[3][3] = {{'X', '0', 'X'}, {'0', 'X', '0'}, {'0', 'X', 'X'}};

Array x

Column		
0	1	2

Row
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

		0	1	2
Row	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.

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 $\times$ size 2 $\times$ … $\times$ 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.

STORING A STRING IN AN ARRAY OF CHARACTERS

The previous syntax display shows that individual characters can be stored in an array by writing each character in the initialization list. If the list is long, this can be done more easily by using a string instead of an initialization list.

char vowels[] = "Hello World";

EXAMPLE (TWO-DIMENSIONAL ARRAY)

```c
//Program to display the transpose of given 2x2 matrix(2D array)

#include <stdio.h>

int main()
{

        int matrix[2][2], transpose[2][2], row, col;

        // Storing elements of the matrix
        printf("\nEnter elements of matrix:\n");
        for(row=0; row<2; row++)
          for(col=0; col<2; col++)
          {
            printf("Enter element a[%d][%d]: ",row,col);
            scanf("%d", &matrix[row][col]);
          }

        // Displaying the matrix[][]
        printf("\nEntered Matrix: \n");
        for(row=0; row<2; row++)
          for(col=0; col<2; col++)
          {
            printf("%d  ", matrix[row][col]);
            if (col == 1)
              printf("\n\n");
          }

        // Finding the transpose of matrix
        for(row=0; row<2; row++)
          for(col=0; col<2; col++)
          {
            transpose[col][row] = matrix[row][col];
```

```
        }

        // Displaying the transpose of matrix
        printf("\nTranspose of Matrix:\n");
        for(row=0; row<2; row++)
          for(col=0; col<2; col++)
          {
             printf("%d ",transpose[row][col]);
             if(col==1)
                printf("\n\n");
          }

   return 0;
}
```

```
Enter elements of matrix:
Enter element a[0][0]: 1
Enter element a[0][1]: 2
Enter element a[1][0]: 3
Enter element a[1][1]: 4

Entered Matrix:
1   2

3   4


Transpose of Matrix:
1   3

2   4
```

Review Loops with Counter

For the loops, three Steps are required in Loop with loop Control Variables (LCV)
● Initialize LCV before the loop starts
● Test Condition using LCV
● Update LCV in body of the loop

Nested For Loop

● A for loop can contain any kind of statement in its body, including another for loop.
● The inner loop must have a different name for its loop counter variable so that it will not conflict with the outer loop.
● Nested loop: Loops placed inside one another, creating a loop of loops.

IMPLEMENTATION IN C

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

for (int i=1; i<=5;i++)
{
   for (int j=1;j<=i;j++)
   {
    printf("*");
   }
printf("\n");
 }
```

```
for ( expr1a; expr2a; expr3a )
{
                   :
  for ( expr1b; expr2b; expr3b )
  {
                   :
  }
                   :
}
```

Nested While Loop

Using While loop within while loops is said to be **Nested while loop**.

In nested while loop one or more statements are included in the body of the loop.

IMPLEMENTATION IN C

```
#include<stdio.h>
int main()
{
  int r,c,s;
  r=1;
  while(r<=5)  /*outer loop*/
  {
  c=1;
  while(c<=2)  /*inner loop*/
  {
   s=r+c;
   printf("r=%d c=%d sum=%d\n",r,c,s);
   c++;
  }
 printf("\n");
 r++;
}}
```

```
while (expr1)

    {

    :
            while (expr2)

                {
                :
                Update expr2;
        }

        :

                update expr1;
    }
```

Nested Do While Loop

Using do While loop within do while loops is said to be **Nested while loop**.

In nested do while loop one or more statements are included in the body of the loop.

IMPLEMENTATION IN C

```
#include<stdio.h>
int main()
{
  int i=1,j=0,sum;
  do
  {
   sum=0;
   do
   {
    sum=sum+j;
    printf("%d",j);
    j++;
    if(j<=i)
    {
      printf("+");
     }
   }
   while(j<=i);
   printf("=%d\n",sum);
   j=1;
   i ++;
   }
   while(i<=10);}
```

```
do

    {

    :
            do

                {
                :
                Update expr;
        }while(expr);

        :

                update expr;
    }while(expr);
```