

MODULE 7 DERIVED DATA TYPE - ARRAY

My Training Period: hours

Note: Another important topic in C and C++.

Abilities

- Able to understand, use and create an array.
- Able to understand and use array and function.
- Able to understand and use array, function and a pointer.
- Able to understand and use array and string.

2.1 Introduction And Definition

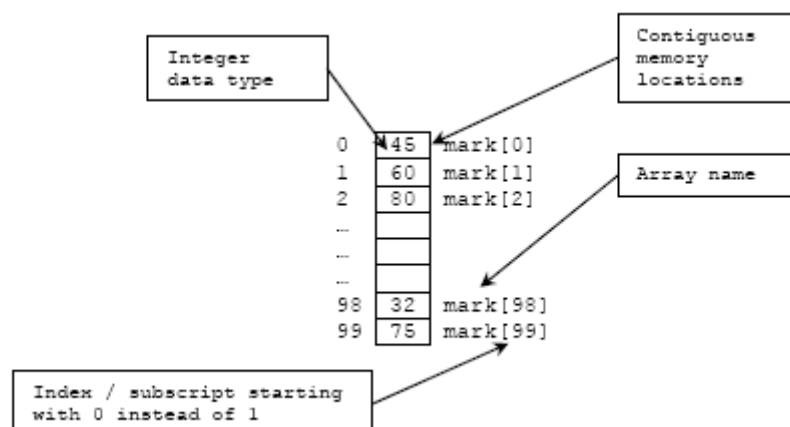
- Up till this Module we were introduced with simple data types that suit to simple applications. For aggregate data, with same type of element we can use Array.
- An array in C/C++ is a collection of related data elements of the same type that are referenced by a common name. Generally, it is just another data type, aggregate data type.
- All the elements of an array occupy a set of contiguous memory locations and by using an **index** or **subscript** we can identify each element.
- For example, instead of declaring `mark1, mark2, ..., markN` to store and manipulate a set of marks obtained by the students in certain courses, we could declare a single array variable named `mark` and use an index, such as `j`, to refer to each element in `mark`. This absolutely has simplified our declaration of the variables.
- Hence, `mark[j]` would refer to the `j`th element in the array `mark`. Thus by changing the value of `j`, we could refer to any element in the array, so it simplifies our declaration.
- For example, if we have 100 list of marks of integer type, we will declare it as follows:

```
int mark1, mark2, mark3, ... , mark100;
```

- If we have 100 marks to be stored, you can imagine how long we have to write the declaration part by using normal variable declaration?
- By using an array, we just declare like this:

```
int mark[100];
```

- This will reserve 100 contiguous/sequential memory locations for storing the integer data type.
- Graphically can be depicted as follows:



7.2 One Dimensional Array: Declaration

- Dimension refers to the array size that is how big the array is. A single dimensional array declaration has the following form:

```
array_element_data_type array_name[array_size];
```

- Here, `array_element_data_type` declares the base type of the array, which is the type of each element in the array. `array_size` defines how many elements the array will hold. `array_name` is any valid C/C++ identifier name that obeys the same rule for the identifier naming.
- For example, to declare an array of 20 characters, named `character`, we could use:

```
char character[20];
```

- Can be depicted as follows:

'A'	character[0]
'h'	character[1]
...	character[2]
...	...
...	...
'I'	character[18]
'l'	character[19]

- In this statement, the array `character` can store up to 20 characters with the first character occupying location `character[0]` and the last character occupying `character[19]`. Note that the index runs from 0 to 19. In C/C++, an index always starts from **0** and ends with (**array size-1**). So, notice the difference between the array **size** and **subscript** terms.
- Examples of the one-dimensional array declarations:

```
int x[20], y[50];
float price[10], yield;
char letter[70];
```

- The first example declares two arrays named `x` and `y` of type `int`. Array `x` can store up to 20 integer numbers while `y` can store up to 50 numbers. The second line declares the array `price` of type `float`. It can store up to 10 floating-point values.
- The third one declares the array `letter` of type `char`. It can store a string up to 69 characters. (Why 69? Remember, a string has a null character (`\0`) at the end, so we must reserve for it.)
- Just like ordinary variables, arrays of the same data type can be declared on the same line. They can also be mixed with ordinary variables of the same data type like in the second line together with `yield`.

7.3 Array Initialization

- An array may be initialized at the time of its declaration, which means to give initial values to an array. Initialization of an array may takes the following form:

```
type array_name[size] = {value_list};
```

- For examples:

```
int id[7] = {1, 2, 3, 4, 5, 6, 7};
float x[5] = {5.6, 5.7, 5.8, 5.9, 6.1};
char vowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};
```

- The first line declares an integer array `id` and it immediately assigns the values 1, 2, 3, ..., 7 to `id[0]`, `id[1]`, `id[2]`, ..., `id[6]`.
- In the second line assigns the values 5.6 to `x[0]`, 5.7 to `x[1]`, and so on.
- Similarly the third line assigns the characters 'a' to `vowel[0]`, 'e' to `vowel[1]`, and so on. Note again, for characters we must use the single apostrophe (') to enclose them. Also, the last character in the array `vowel` is the NULL character (`\0`).
- Initialization of an array of type `char` for holding strings may takes the following form:

```
char array_name[size] = "string_lateral_constant";
```

- For example, the array `vowel` in the above example could have been written more compactly as follows:

```
char vowel[6] = "aeiou";
```

- When the value assigned to a character array is a string (which must be enclosed in double quotes), the compiler automatically supplies the NULL character but we still have to reserve one extra place for the NULL.

1.1 Array And Function: Passing One Dimensional Arrays To Function

- A function can receive the address of an array by using:
 1. A pointer.
 2. A sized array (dimension is explicitly stated), e.g. `s[20]` or
 3. An unsized array (dimension is not stated), e.g. `p[]`.
- For example, to receive an array named `x` of type `float` in functions, we may declare any one of the following:

```
// Pointers, will be explained in another Module
int myfunction(float *x)
// Sized array
char yourfunction(float x[5])
// Unsized array
void ourfunction(float x[])
```

- But you will see later, the second and third methods not used in practice.
- The following program segment illustrates the passing of an array address to a function using a pointer.
- Here, the memory address of `x` is passed to the parameter `pter`, a pointer. Remember this; **an array name (without the size) is the pointer to the first array's element**. We will discuss this in more detail in another Module.

```
// function prototype
void func(float *);

int main()
{
    float x[5];

    // an array name (without the bracket) is
    // the pointer to the first array element
    func(x);
    return 0;
}

// function definition
void func(float *pter)
{ return; }
```

1.2 Array Manipulation: How to use an array and what array can do?

1.2.1 Accessing Array's Element

- The following program example declares and initializes the array named `y` of type `int`. It uses a `for` loop with index `i` to access the successive elements in `y`. For each loop iteration, the value accessed is added to the variable `total`, which is finally displayed. Note that the loop index `i` run from 0 to 6, not 1 to 7. Also, note that the array size `n` is declared in the `#define` statement.

```
//Program to find the total of all the elements in array y
#include <iostream.h>
#include <stdlib.h>
//replace every n occurrences with 7
#define n 7

int main()
{
```

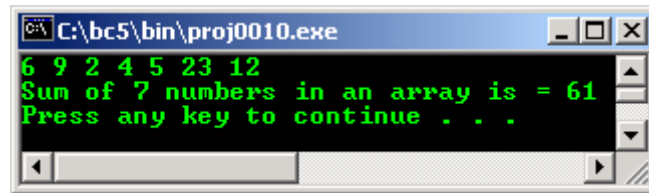
```

int i, total = 0, y[n] = {6,9,2,4,5,23,12};

for (i=0; i<n; i++)
{
    //display the array contents...
    cout<<y[i]<<" ";
    //do the summing up...
    total = total + y[i];
}
//display the result...
cout<<"\nSum of 7 numbers in an array is = "<<total<<endl;
system("pause");
return 0;
}

```

Output :



- You can see that the for loop is very useful when you wish to loop or iterate through every element of an array.
- The next program example, accomplishes the same task as the previous one. By using `get_total()` function, the `main()` passes the first memory address (pointed by the pointer) of an array, `y` and its size, `n` to the function `get_total()` which then computes and returns the total, `total`. Note the function prototype:

```
int get_total(int*, int)
```

- Study the program and the output.

```

//program to find the total values of an
//array y by passing an array to a function
//using pointer
#include <iostream.h>
#include <stdlib.h>
#define n 7

//function prototype
int get_total(int*, int);

int main()
{
    int total, y[n]={6,9,2,4,5,23,12};

    cout<<"\nCalling function get_total(y, n),";
    cout<<"\nBy bringing along the value of y, an array";
    cout<<"\nfirst address and n = 7, an array size.";
    cout<<"\nAn array name, is the pointer to the";
    cout<<"\n1st element of an array\n\n";

    //function call, pass along the pointer to the first
    //array element and the array size, and the
    //return result assign to variable total
    total = get_total(y, n);

    cout<<"\nSum of the 7 array elements is "<<total<<endl;
    system("pause");
    return 0;
}

//Function definition
int get_total(int *ptr, int x)
{
    int i, total = 0;
    //do the looping for array elements...
    for(i=0; i<x; i++)
    {
        //displays the array content, pointed by pointer...

```

```

        cout<<*(ptr+i)<<" ";
        //do the summing up of the array elements...
        total += *(ptr+i); //total=total + *(ptr+i);
    }
    //return the result to the calling program...
    return total;
}

```

Output:

```

C:\bc5\bin\proj0010.exe
Calling function get_total(y, n),
By bringing along the value of y, an array
first address and n = 7, an array size.
An array name, is the pointer to the
1st element of an array

6 9 2 4 5 23 12
Sum of the 7 array elements is 61
Press any key to continue . . .

```

- C++ program example compiled using g++ (for more information refers to [Module000](#)).

```

/*****gccarray.C or gccarray.cpp*****/
/*****FeDoRa 3, g++ x.x.x*****/
//program to find the total values of an
//array y by passing an array to a function
//using pointer
#include <iostream>
#define n 7
using namespace std;

//function prototype
int get_total(int*, int);

int main()
{
    int total, y[n]={6,9,2,4,5,23,12};

    cout<<"\nCalling function get_total(y, n),";
    cout<<"\nBy bringing along the value of y, an array";
    cout<<"\nfirst address and n = 7, an array size.";
    cout<<"\nAn array name, is the pointer to the";
    cout<<"\n1st element of an array\n\n";

    //function call, pass along the pointer to the first
    //array element and the array size, and the
    //return result assign to variable total
    total = get_total(y, n);

    cout<<"\nSum of the 7 array elements is "<<total<<endl;
    return 0;
}

//Function definition
int get_total(int *ptr, int x)
{
    int i, total = 0;
    //do the looping for array elements...
    for(i=0; i<x; i++)
    {
        //displays the array content, pointed by pointer...
        cout<<*(ptr+i)<<" ";
        //do the summing up of the array elements...
        total += *(ptr+i); //total=total + *(ptr+i);
    }
    //return the result to the calling program...
    return total;
}

```

```

[bodo@bakawali ~]$ g++ gccarray.C -o gccarray
[bodo@bakawali ~]$ ./gccarray

```

```
Calling function get_total(y, n),
By bringing along the value of y, an array
first address and n = 7, an array size.
An array name, is the pointer to the
1st element of an array
```

```
6 9 2 4 5 23 12
Sum of the 7 array elements is 61
[bodo@bakawali ~]$
```

7.5.2 Searching For A Value

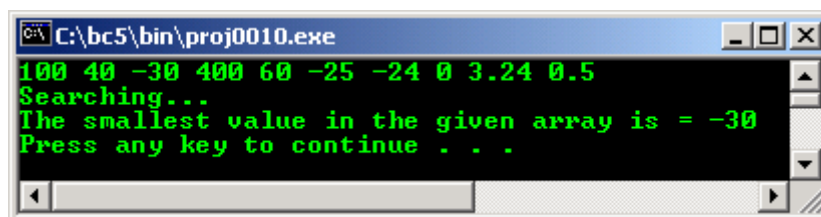
- To search an array for a value, you can use a `for` loop within the `if` statement. In other words, the program looks at each element in the array and checks if it matches the given value in the `if` expression.
- The following example, finds the smallest element in the array named `balance`. First, it assumes that the smallest value is in `balance[0]` and assigns it to the variable `small`.
- Then it compares `small` with the rest of the values in `balance`, one at a time.

```
//program to find the smallest number in an array named balance,
//simple search function
#include <iostream.h>
#include <stdlib.h>
#define n 10

int main()
{
    int i;
    float small, balance[n]={100.00,40.00,-30.00,400.00,60.00,-25.00,-
24.00,0.00,3.24,0.50};
    small = balance[0];
    //loop for displaying array content...
    for(i=0; i<n; i++)
        cout<<balance[i]<<" ";

    //Another loop do the array element comparing...
    for(i=1; i<n; i++) //check until condition i=n
    {
        if(small > balance[i])
            small = balance[i];
    }
    cout<<"\nSearching..."<<endl;
    //display the result...
    cout<<"The smallest value in the given array is = "<<small<<endl;
    system("pause");
    return 0;
}
```

Output:



- When an element is smaller than the current value contained in `small`, it is assigned to `small`. The process finally places the smallest array element in `small`.
- Study the program and the output.

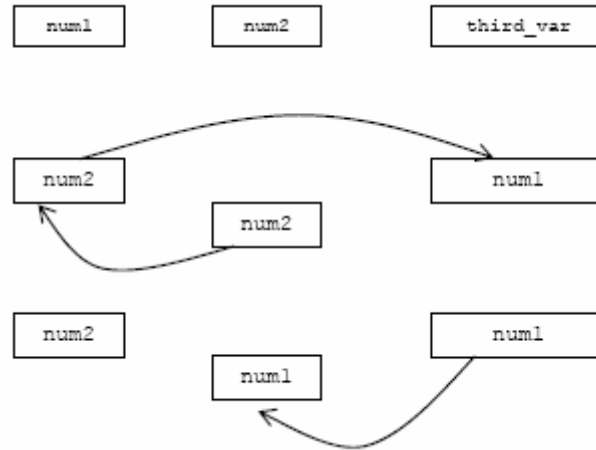
1.1.1 Exchanging Values of Variables

- It is sometimes necessary to shuffle the order of the elements of an array.
- Sorting arrays calls for the values of some elements to be exchanged. It would therefore be helpful to first learn the technique of swapping variables.
- How would you swap the values of the variables, let say, `num1` and `num2` (that is exchanging the value of `num1` in `num2`)?

- You must use a third variable as in the following example:

```
//assign num1 to third_var
third_var = num1;
//then assigns num2 to num1
num1 = num2;
//finally assigns third_var to num2
num2 = third_var;
```

- The process can be illustrated as shown below.



1.1.2 Sorting Variables

- Sorting is defined as arranging data in a certain order, is a very common activity in data processing. Many algorithms (techniques) are available to perform sorting.
- One sorting algorithm, which is quite simple to understand, but not necessarily the best, is given in the following program example. It sorts a list of integers in ascending order of magnitude by using an array.
- For more algorithms implemented using [Template](#) of STL (C++), you can refer to Tutorial #5.

```
//Simple sorting program that sort a list of n
//integer numbers, entered by the user (ascending)
#include <iostream.h>
#include <stdlib.h>
#define  maxsize  100

int main()
{
    int temp, i, j, n, list[maxsize];

    cout<<"\n--You are prompted to enter your list size.--";
    cout<<"\n--Then, for your list size, you are prompted to enter--";
    cout<<"\n--the element of your list.--";
    cout<<"\n--Finally your list will be sorted ascending!!!--\n";

    //get the list size...
    cout<<"\nEnter your list size: ";
    cin>>n;

    //prompting the data from user store in the list...
    for(i=0; i<n; i++)
    {
        cout<<"Enter list's element #"<<i<<"-->";
        cin>>list[i];
    }

    //do the sorting...
    for(i=0; i<n-1; i++)
    {
        for(j=i+1; j<n; j++)
            if(list[i] > list[j])
            {
                //These three lines swap the elements
                //list[i] and list[j].
                temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
    }
}
```

```

        list[i] = list[j];
        list[j] = temp;
    }
    cout<<"\nSorted list, ascending:  ";

    for(i=0; i<n; i++)
        cout<<" "<<list[i];
        cout<<endl;
    system("pause");
    return 0;
}

```

Output:

```

C:\bc5\bin\proj0010.exe
--You are prompted to enter your list size.--
--Then, for your list size, you are prompted to enter--
--the element of your list.--
--Finally your list will be sorted ascending!!!--

Enter your list size: 10
Enter list's element #0-->23
Enter list's element #1-->12
Enter list's element #2-->7
Enter list's element #3-->45
Enter list's element #4-->99
Enter list's element #5-->15
Enter list's element #6-->41
Enter list's element #7-->57
Enter list's element #8-->88
Enter list's element #9-->91

Sorted list, ascending: 7 12 15 23 41 45 57 88 91 99
Press any key to continue . . .

```

1.2 Two Dimensional Arrays

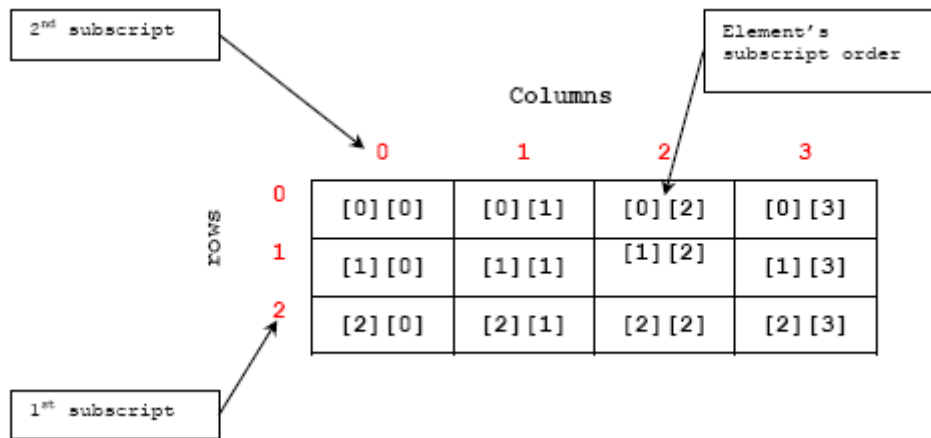
- Some data fit better in a table with several rows and columns. This can be constructed by using two-dimensional arrays.
- A two dimensional array has two subscripts/indexes. The **first subscript refers to the row**, and the **second**, to the **column**. Its declaration has the following form:

```
data_type array_name[1st dimension size][2nd dimension size];
```

- For examples:

```
int    x[3][4];
float  matrix[20][25];
```

- The first line declares x as an integer array with 3 rows and 4 columns and the second line declares a matrix as a floating-point array with 20 rows and 25 columns.
- Descriptively, int x[3][4] can be depicted as follows:



- You can see that for [3] [4] two-dimension array size; the total array size (the total array elements) is equal to 12. Hence:

For n rows and m columns, the total size equal to mn

- The item list is read starting from the first row from left to right, and then goes to the next row and so on.
- A set of string s can be stored in a two-dimensional character array with the left index specifying the number of strings and the right index specifying the maximum length of each string.
- For example, to store a list of 3 names with a maximum length of 10 characters in each name, we can declare:

```
char name[4][10];
//can store 4 names, each is 10 characters long
```

- Just like the one-dimensional array, a two dimensional array can also be initialized. For example, the previous first array declaration can be rewritten along with initial assignments in any of the following ways:

```
int x[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

- Or

```
int x[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

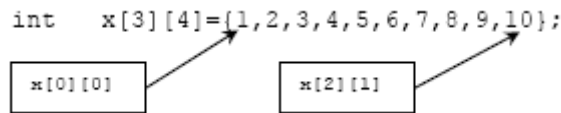
- The results of the initial assignments in both cases are as follows:

```
x[0][0]=1      x[0][1]=2      x[0][2]=3      x[0][3]=4
x[1][0]=5      x[1][1]=6      x[1][2]=7      x[1][3]=8
x[2][0]=9      x[2][1]=10     x[2][2]=11     x[2][3]=12
```

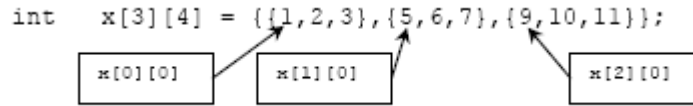
- Notice the same subscript for the rows and columns; it is in the diagonal line.
- You can show how the rows are filled during its initialization. For example, the array named x can be declared as follows:

```
int x[3][4] = { {1,2,3,4},
                {5,6,7,8},
                {9,10,11,12}
              };
```

- If the number of values given is insufficient to fill in the whole array, an initial value of zero will be assigned to all those locations, which are not given initial values explicitly.
- For example:



- So, an initial value of zero will be assigned to `x[2][2]` and `x[2][3]`. Similarly, in declaration:



- An initial value of zero will be assigned to `x[0][3]`, `x[1][3]` and `x[2][3]`. You can fill the whole array with zeroes by using:

```
int x[3][4]={0}; //all array elements will be 0
```

- In memory, despite their table-form arrangement, the elements of a two-dimensional array are stored sequentially, that mean one after another contiguously.
- An array of string can also be initialized. For example, in the following declaration:

```
char name[4][10] = {"Sally", "Joyce", "Lisa", "Alice"};
```

- The values assigned are as follows:

```
name[0] = "Sally"      name[1] = "Joyce"
name[2] = "Lisa"      name[3] = "Alice"
```

- Note that the second subscript here is unnecessary and therefore omitted.

1.3 Two-Dimensional Array Manipulation

- The following example prints the 3 x 3 array's subscript and their element.

```
//Printing 3x3 array's subscript and their element
#include <iostream.h>
#include <stdlib.h>
#define m 3
#define n 3

int main()
{
    int i, j;
    int x[m][n]={{10,25,33}, {21,32,43},{20,42,51}};

    cout<<"\n3x3 arrays' subscripts and\n";
    cout<<"their respective elements\n";
    cout<<"-----\n";

    //outer for loop, reading the row by row...
    for(i=0; i<m; i++)
        //inner loop, for every row, read every column by column...
        for(j=0; j<n; j++)
            cout<<"["<<i<<"]"<<"["<<j<<"]"<<"="<<x[i][j]<<"\n";
    system("pause");
    return 0;
}
```

Output :

```

C:\bc5\bin\proj0010.exe
3x3 arrays' subscripts and
their respective elements
-----
[0][0]=10
[0][1]=25
[0][2]=33
[1][0]=21
[1][1]=32
[1][2]=43
[2][0]=20
[2][1]=42
[2][2]=51
Press any key to continue . . .

```

- The following program example illustrates the use of two-dimensional arrays. This program calculates the average of all the elements in the integer array named x. The program uses two nested for loops.
- The outer loop with index i provides the row subscript. The nested for loops therefore accesses each element of the array and the inner loop with index j provides the column subscript.

```

//using two-dimensional array to compute the
//average of all the elements in array named x
#include <iostream.h>
#include <stdlib.h>
#define m 4
#define n 5

int main()
{
    int i, j, total = 0;
    //an 4x5 or [4][5] array variable...
    int q[m][n]={{4,5,6,2,12},{10,25,33,22,11},
                 {21,32,43,54,65},{3,2,1,5,6}};

    float average;

    //outer for loop, read row by row...
    for(i=0; i<m; i++)
        //inner for loop, for every row, read column by column
        for(j=0; j<n; j++)
            //get the summation of the array elements.
            {
                //display the array...
                cout<<"q["<<i<<"]["<<j<<"] = "<<q[i][j]<<endl;
                total=total + q[i][j];
            }
    //calculate the average
    //simple typecast from int to float...
    average = (float)total/(float) (m*n);

    cout<<"\nThis program will calculate the average of the";
    cout<<"\n4 x 5 array, which means the sum of the";
    cout<<"\narray's element, divide the number of the";
    cout<<"\narray's element....";
    cout<<"\nProcessing.... PLEASE WAIT\n";

    //display the average
    cout<<"Average = "<<total<<"/"<<m*n<<endl;
    cout<<"\nThe Average = "<<average<<endl;
    system("pause");
    return 0;
}

```

Output :

```

C:\bc5\bin\proj0010.exe
q[0][1] = 5
q[0][2] = 6
q[0][3] = 2
q[0][4] = 12
q[1][0] = 10
q[1][1] = 25
q[1][2] = 33
q[1][3] = 22
q[1][4] = 11
q[2][0] = 21
q[2][1] = 32
q[2][2] = 43
q[2][3] = 54
q[2][4] = 65
q[3][0] = 3
q[3][1] = 2
q[3][2] = 1
q[3][3] = 5
q[3][4] = 6

This program will calculate the average of the
4 x 5 array, which means the sum of the
array's element, divide the number of the
array's element....
Processing.... PLEASE WAIT
Average = 362/20

The Average = 18.1
Press any key to continue . . .

```

- Study the program and the output.
- The next example computes the square root of the sum of the squares of all the positive elements in array named x. It uses the header file `math.h` since it contains the mathematical functions `pow()` (for taking the power of a number) and `sqrt()` (for taking the square root of a number).

```

//program to compute the square root of the sum
//of the squares of all the elements in array x
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#define m 4
#define n 5

int main()
{
    int i, j;
    int x[m][n]={{4,5,6,2,12},{10,25,33,22,11},
                {21,32,43,54,65},{3,2,1,5,6}};

    float sum2, result;

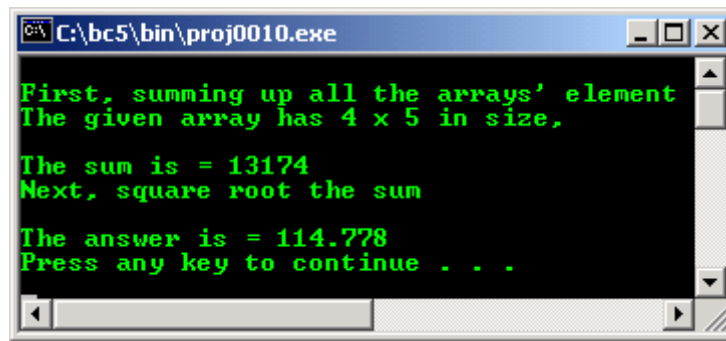
    //outer for loop, read row by row...
    for(i=0; i<m; i++)
        { //inner for loop, for every row, read column by column
            for(j=0; j<n; j++)
                {
                    //set some condition here to avoid divides by 0...
                    if(x[i][j]>0)
                        //do the square of the array elements and then sum up...
                        sum2 = sum2 + pow(x[i][j], 2);
                }
            //assign the result to variable result...
            //do the square root of the previous result....
            result = sqrt(sum2);
        }

    //some story and printing the result...
    cout<<"\nFirst, summing up all the arrays' element";
    cout<<"\nThe given array has 4 x 5 in size,\n";
    cout<<"\nThe sum is = "<<sum2;
    cout<<"\nNext, square root the sum\n";
    cout<<"\nThe answer is = "<<result<<"\n";
    system("pause");
    return 0;
}

```

```
}
```

Output:



- Study the program and the output.
- The following program example illustrates the use of three nested for loops. The program multiplies matrix x and y and stores the resulting matrix product xy in matrix z. Both x and y must be compatible for multiplication that means, the number of columns of x must be equal to the number of rows of y.

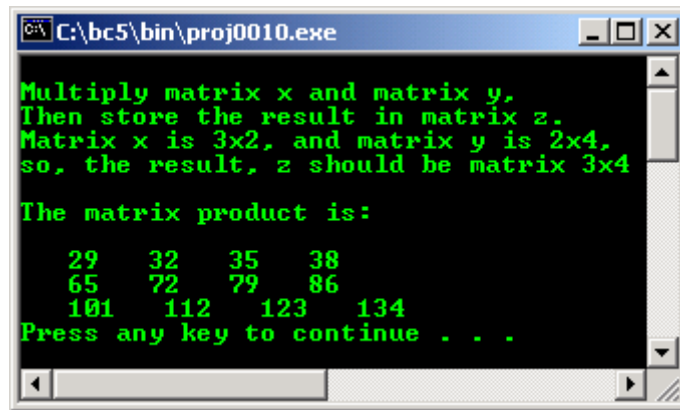
```
//multiplication of the matrix x and matrix
//y and stores the result in matrix z
#include <iostream.h>
#include <stdlib.h>
#define m 3
#define c 2
#define n 4

int main()
{
    int i, j, k;
    //first matrix...
    int x[m][c] = {{1,2},{3,4},{5,6}};
    //second matrix...
    int y[c][n] = {{7,8,9,10},{11,12,13,14}};
    //for storing the matrix product result...
    int z[m][n];

    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
        {
            z[i][j] = 0;
            for(k=0; k<c; k++)
                //same as z[i][j] = z[i][j] + x[i][k] * y[k][j];
                z[i][j] += x[i][k] * y[k][j];
        }
    cout<<"\nMultiply matrix x and matrix y,";
    cout<<"\nThen store the result in matrix z.";
    cout<<"\nMatrix x is 3x2, and matrix y is 2x4,";
    cout<<"\nso, the result, z should be matrix 3x4\n";
    cout<<"\nThe matrix product is: \n";

    for (i=0; i<m; i++)
    {
        cout<<"\n";
        for(j=0; j<n; j++)
            //display the result...
            cout<<" "<<z[i][j];
    }
    cout<<endl;
    system("pause");
    return 0;
}
```

Output:



- Study the program and the output.

1.4 Multidimensional Arrays

- When an array has more than one dimension, we call it a multidimensional array.
- We have already looked at multidimensional arrays with two dimensions. The declaration and manipulation of other multidimensional arrays in C/C++ are quite similar to that of the two dimensional array.
- The declaration takes the following form:

```
Data_type name[size1][size2]...[sizeN];
```

- For example:

```
int y[4][5][3];
```

- Declares a 3-dimensional array with a depth of 4, 5 rows and 3 columns.
- There are no limitations on the dimension of the arrays, but the dimension more than two dimensional arrays are rarely used because of the complexity and code readability.

-----o0o-----
---www.tenouk.com---

Program Examples

Example #1

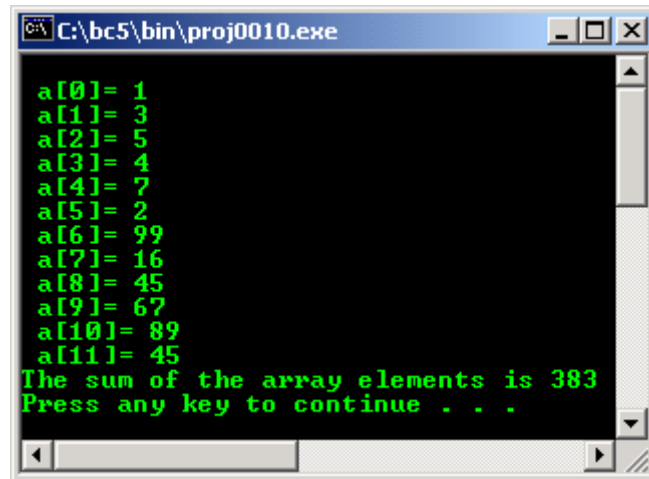
```
//Compute the sum of the elements of an array
#include <stdlib.h>
#include <stdio.h>
#define SIZE 12

int main()
{
    //declare and initialize the array named a
    //with size SIZE
    int a[SIZE] = {1,3,5,4,7,2,99,16,45,67,89,45};
    //declare two normal variables
    int i, total = 0;

    //do the loop for the array...
    for(i = 0; i <= (SIZE-1); i++)
    {
        //display the array and its element...
        printf("\n a[%d]= %d", i, a[i]);
        //total up the array
        //total = total + a[i]
        total += a[i];
    }

    printf("\nThe sum of the array elements is %d\n", total);
    system("pause");
    return 0;
}
```

Output:



```
C:\bc5\bin\proj0010.exe
a[0]= 1
a[1]= 3
a[2]= 5
a[3]= 4
a[4]= 7
a[5]= 2
a[6]= 99
a[7]= 16
a[8]= 45
a[9]= 67
a[10]= 89
a[11]= 45
The sum of the array elements is 383
Press any key to continue . . .
```

Example #2

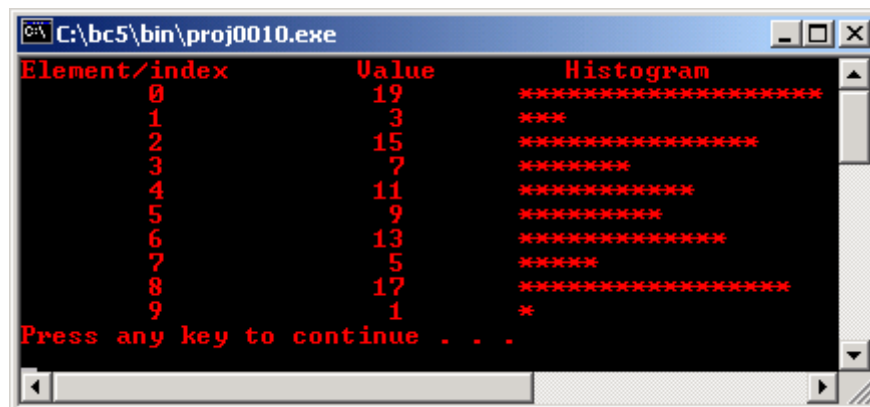
```
//Printing simple histogram
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int main()
{
    //declare and initialize an array named n
    //with size SIZE...
    int n[SIZE] = {19, 3, 15, 7, 11, 9, 13, 5, 17, 1};
    int i, j;

    //display the table header...
    printf("%s%13s%17s\n", "Element/index", "Value", "Histogram");

    //do the iteration...
    //outer for loop, read row by row...
    for(i=0; i <= (SIZE-1); i++)
    {
        printf("%9d%15d      ", i, n[i]);
        //inner for loop, for every row, read column
        //by column and print the bar...
        for(j = 1; j<= n[i]; j++)
            //print bar...repeat...
            printf("*");
        //go to new line for new row...repeats...
        printf("\n");
    }
    system("pause");
    return 0;
}
```

Output:



```
C:\bc5\bin\proj0010.exe
Element/index      Value      Histogram
0                 19      *****
1                  3         ***
2                 15      *****
3                  7         *****
4                 11      *****
5                  9         *****
6                 13      *****
7                  5         *****
8                 17      *****
9                  1          *
Press any key to continue . . .
```

Example #3

```
//Sorting an array values into ascending order
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

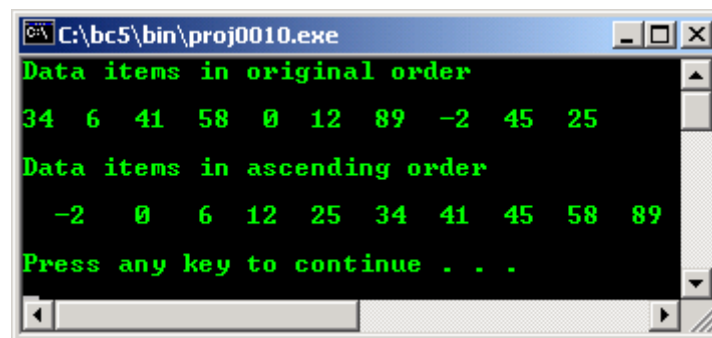
int main()
{
    int a[SIZE] = {34,6,41,58,0,12,89,-2,45,25};
    int i, pass, hold;

    printf("Data items in original order\n\n");
    //displaying the original array...
    for(i=0; i<=SIZE - 1; i++)
        printf("%d  ", a[i]);

    //-----do the sorting...ascending-----
    //for every array elements do this...
    for(pass = 1; pass <= (SIZE-1); pass++)
        //for every 2 array elements comparison do
        //the comparison and swap...
        for(i = 0; i <= (SIZE-2); i++)
            //set the condition...
            if(a[i] > a[i + 1])
            {
                //put the a[i] in temporary variable hold...
                hold = a[i];
                //put the a[i + 1] in a[i]
                a[i] = a[i + 1];
                //put the hold in a[i + 1], one swapping is
                //completed...and repeat for other elements...
                a[i + 1] = hold;
            }

    printf("\n\nData items in ascending order\n\n");
    //display the new ordered list...
    for (i=0; i <= (SIZE-1); i++)
        printf("%4d", a[i]);
        printf("\n\n");
    system("pause");
    return 0;
}
```

Output:



- By changing the following code in the above program, recompile and re run the program. You will get the descending order.

```
if(a[i] > a[i + 1]) to if(a[i] < a[i + 1])
```

- The following is the output.


```

C:\d:\conversion\Debug\conversion.exe
Data items in original order
34 6 41 58 0 12 89 -2 45 25
Data items in ascending order
89 58 45 41 34 25 12 6 0 -2
Press any key to continue . . .

```

Example #4

```

//Initializing multidimensional arrays
//and function
#include <stdio.h>
#include <stdlib.h>

//function prototype
void printArray(int a[][3]);

int main()
{
    //declare 3 array with initial values...
    int array1[2][3] = {{1,2,3}, {4,5,6}},
        array2[2][3] = {{1,2,3},{4,5}},
        array3[2][3] = {{1,2}, {4}};

    printf("Element values in array1 by row are: \n");
    //first time function call
    printArray(array1);
    printf("\nElement values in array2 by row are: \n");

    //second time function call
    printArray(array2);
    printf("\nElement values in array3 by row are:\n");

    //third time function call
    printArray(array3);
    printf("\nNOTICE THE DEFAULT VALUE 0...\n");
    system("pause");
    return 0;
}

//function definition, passing an array to function
void printArray(int a[][3])
{
    int i, j;
    //outer for loop, read row by row...
    for(i = 0; i <= 1; i++)
    {
        //inner for loop, for every row, read column by column...
        for(j=0; j<= 2; j++)
        {
            printf("[%d][%d] = %d ", i, j, a[i][j]);
        }
        printf("\n");
    }
}

```

Output:

```

C:\bc5\bin\proj0010.exe
Element values in array1 by row are:
[0][0] = 1 [0][1] = 2 [0][2] = 3
[1][0] = 4 [1][1] = 5 [1][2] = 6

Element values in array2 by row are:
[0][0] = 1 [0][1] = 2 [0][2] = 3
[1][0] = 4 [1][1] = 5 [1][2] = 0

Element values in array3 by row are:
[0][0] = 1 [0][1] = 2 [0][2] = 0
[1][0] = 4 [1][1] = 0 [1][2] = 0

NOTICE THE DEFAULT VALUE 0...
Press any key to continue . . .

```

Example #5

```

//program will sort a list of
//a strings entered by the user
#include <iostream.h>
#include <stdlib.h>
#include <string.h>

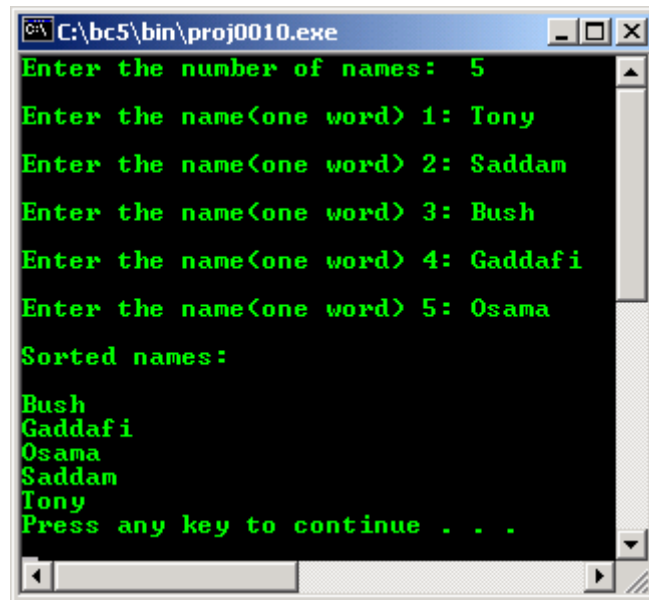
int main()
{
    //declare two arrays named tname with 1-Dimension
    //and name with 2-Dimension
    char tname[20], name[20][20];
    //normal variables...
    int i, j, n;

    cout<<"Enter the number of names: ";
    cin>>n;
    //outer loop for counter...
    for(i=0; i<n; i++)
    {
        cout<<"\nEnter the name(one word) "<<(i+1)<<": ";
        cin>>name[i];
    }

    //inner for loop, read row by row set outer for loop...
    for(i=0; i<n-1; i++)
    //innermost for loop, read column by column of the characters...
    for(j = i+1; j<n; j++)
    //set the condition...
    //strcmp - compare the string standard library function
    //do the sorting...
    if(strcmp(name[i], name[j])>0)
    {
        //strcpy - copy the strings...
        //compare and swap...
        strcpy(tname, name[i]);
        strcpy(name[i], name[j]);
        strcpy(name[j], tname);
    }
    cout<<"\nSorted names:\n";
    for (i =0; i<n; i++)
    cout<<"\n"<<name[i];
    cout<<endl;
    system("pause");
    return 0;
}

```

Sample output:



```
C:\bc5\bin\proj0010.exe
Enter the number of names: 5
Enter the name(one word) 1: Tony
Enter the name(one word) 2: Saddam
Enter the name(one word) 3: Bush
Enter the name(one word) 4: Gaddafi
Enter the name(one word) 5: Osama
Sorted names:
Bush
Gaddafi
Osama
Saddam
Tony
Press any key to continue . . .
```

- Program example compiled using VC++/VC++ .Net.

```
//Sorting array values into ascending order
#include <stdio>
#define SIZE 10

int main()
{
    int a[SIZE] = {-4,6,3,-20,0,1,77,-2,42,-10};
    int i, pass, hold;

    printf("Data items in original order\n\n");
    //displaying the original array...
    for(i=0; i<=SIZE - 1; i++)
        printf("%d ", a[i]);

    //-----do the sorting...ascending-----
    //for every array elements do this...
    for(pass = 1; pass <= (SIZE-1); pass++)
        //for every 2 array elements comparison do
        //the comparison and swap...
        for(i = 0; i <= (SIZE-2); i++)
            //set the condition...
            if(a[i] > a[i + 1])
            {
                //put the a[i] in temporary variable hold...
                hold = a[i];
                //put the a[i + 1] in a[i]
                a[i] = a[i + 1];
                //put the hold in a[i + 1], one swapping is
                //completed...and repeats for other elements...
                a[i + 1] = hold;
            }

    printf("\n\nData items in ascending order\n\n");
    //display the new ordered list...
    for(i=0; i <= (SIZE-1); i++)
        printf("%4d", a[i]);
    printf("\n\n");
    return 0;
}
```

Output:

```

C:\g:\vcnetprojek\searchpattern\Debug\s...
Data items in original order
-4 6 3 -20 0 1 77 -2 42 -10
Data items in ascending order
-20 -10 -4 -2 0 1 3 6 42 77
Press any key to continue

```

- In C++, you can use member functions, operators, and classes etc. of the <valarray> [template](#) based header file of the **Standard Template Library** (STL). It is designed for performing high-speed mathematical operations, and optimized for computational performance.
- The class is a one-dimensional smart array that checks subscript references at run time to confirm that they are in bound.
- Also, in C++ many array constructs and manipulations accomplished using routines in Standard Template Library (STL). For further discussion, please refer to Tutorial #4.
- Take care also when you use arrays as buffers for storing strings (NULL terminated). Many standard functions for string operations such as `strcpy()` do not do the bound checking. This will result a buffer overflow.
- For example, consider the following program:

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    //reserve 5 byte of buffer...
    //should allocate 8 bytes = 2 double words,
    //to overflow, need more than 8 bytes...
    //so, if more than 8 characters input by user,
    //there will be access violation, segmentation fault etc
    char mybuffer[5];
    //a prompt how to execute the program...
    if(argc < 2)
    {
        printf("strcpy() NOT executed...\n");
        printf("Syntax: %s <characters>\n", argv[0]);
        exit(0);
    }

    //copy the user input to mybuffer...
    strcpy(mybuffer, argv[1]);
    printf("mybuffer content= %s\n", mybuffer);
    printf("strcpy() executed...\n");
    return 0;
}

```

- The output, when the input is: 12345678 (8 bytes), the program run smoothly.

```

C:\WINNT\system32\cmd.exe
E:\test\buffalo\Debug>buffalo
strcpy() NOT executed...
Syntax: buffalo <characters>

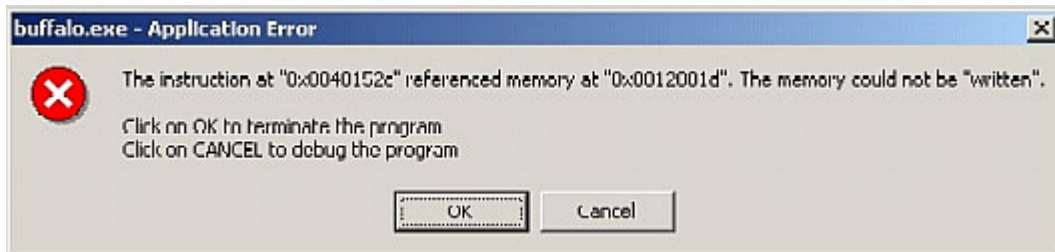
E:\test\buffalo\Debug>buffalo 12345678
mybuffer content= 12345678
strcpy() executed...

E:\test\buffalo\Debug>buffalo 123456789
mybuffer content= 123456789
strcpy() executed...

E:\test\buffalo\Debug>

```

- When the input is: 123456789 (9 bytes), the following will be displayed when compiled with Microsoft Visual C++ 6.0. In Linux the “Segmentation fault” message will be displayed and the program terminates.



- The vulnerability exist because the `mybuffer` could be overflowed if the user input (`argv[1]`) bigger than 8 bytes. Why 8 bytes? For 32 bit (4 bytes) system, we must fill up a double word (32 bits) memory.
- Character (`char`) size is 1 byte, so if we request buffer with 5 bytes, the system will allocate 2 double words (8 bytes). That is why when you input more than 8 bytes, the `mybuffer` will be overflowed!
- The system tries to write beyond the allocated storage (memory) which may be occupied by other important data.
- You may use safer type functions or insert some codes that do the bound checking.
- In real world, the buffer overflow vulnerabilities have been widely exploited by viruses, worms and malicious codes.
- As a final note for this Module, the following are previous program examples compiled using `gcc`.

```

/*****array3.c*****/
/*program to find the smallest number in an array named balance*/
/*simple search function*/
#include <stdio.h>
#define n 7

int main()
{
int i;
int small, balance[n];

/**loop for displaying array content...*/
for(i=0; i<=n; i++)
{
printf("Key in float value, let me ... for you: ");
scanf("%d", &balance[i]);
}

/*printing the element...*/
for(i=0; i<=n; i++)
printf("%d ", balance[i]);

small = balance[0];
/*Another loop do the array element comparing...*/
for(i=1; i<=n; i++) /*check until i=n*/
{
if(small > balance[i])
small = balance[i];
}

printf("\nComparing...");
/*display the result...*/
printf("The smallest value in the given array is = %d \n", small);
return 0;
}

```

```

[bodo@bakawali ~]$ gcc array3.c -o array3
[bodo@bakawali ~]$ ./array3

```

```

Key in float value, let me ... for you: 12
Key in float value, let me ... for you: -21
Key in float value, let me ... for you: 4
Key in float value, let me ... for you: -3
Key in float value, let me ... for you: 0
Key in float value, let me ... for you: 7
Key in float value, let me ... for you: -41

```

```
Key in float value, let me ... for you: 3
12 -21 4 -3 0 7 -41 3
Comparing...The smallest value in the given array is = -41
```

```
/******array1.c******/
/*Simple sorting program that sort a list of n*/
/*integer numbers, entered by the user (ascending)*/
#include <stdio.h>
#define  maxsize  100

int main()
{
int temp, i, j, n, list[maxsize];

printf("\n--You are prompted to enter your list size.--");
printf("\n--Then, for your list size, you are prompted to enter--");
printf("\n--the element of your list.--");
printf("\n--Finally your list will be sorted ascending--\n");

/*get the list size...*/
printf("\nEnter your list size: ");
scanf(" %d", &n);

/*prompting the data from user store in the list...*/
for(i=0; i<n; i++)
{
printf("Enter list's element #i -->", i);
scanf("%d", &list[i]);
}

//do the sorting...
for(i=0; i<n-1; i++)
for(j=i+1; j<n; j++)
if(list[i] > list[j])
{
/*These three lines swap the elements*/
/*list[i] and list[j].*/
temp = list[i];
list[i] = list[j];
list[j] = temp;
}
printf("\nSorted list, ascending:  ");

for(i=0; i<n; i++)
printf(" %d", list[i]);
printf("\n");
return  0;
}
```

```
[bodo@bakawali ~]$ gcc array1.c -o array1
[bodo@bakawali ~]$ ./array1
```

```
--You are prompted to enter your list size.--
--Then, for your list size, you are prompted to enter--
--the element of your list.--
--Finally your list will be sorted ascending--

Enter your list size: 10
Enter list's element #0 -->23
Enter list's element #1 -->14
Enter list's element #2 -->-21
Enter list's element #3 -->-30
Enter list's element #4 -->34
Enter list's element #5 -->25
Enter list's element #6 -->12
Enter list's element #7 -->99
Enter list's element #8 -->100
Enter list's element #9 -->73

Sorted list, ascending:   -30  -21  12  14  23  25  34  73  99  100
```

```
/******array2.c******/
/*Printing 3x3 array's subscript and their element*/
#include <stdio.h>
#define m  3
#define n  3
```

```

int main()
{
int i, j;
int x[m][n];

printf("\n3x3 arrays' subscripts and\n");
printf("their respective elements\n");
printf("-----\n");

    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
        {
printf("Enter int values for ur array lol!: ");
scanf("%d", &x[i][j]);
}
/*outer for loop, reading the row by row...*/
for(i=0; i<m; i++)
/*inner loop, for every row, read every column by column...*/
for(j=0; j<n; j++)
printf("x[%d][%d] = %d\n", i, j, x[i][j]);
return 0;
}

```

```

[bodo@bakawali ~]$ gcc array2.c -o array2
[bodo@bakawali ~]$ ./array2

```

```

3x3 arrays' subscripts and
their respective elements
-----
Enter int values for ur array lol!: 12
Enter int values for ur array lol!: 31
Enter int values for ur array lol!: 45
Enter int values for ur array lol!: -20
Enter int values for ur array lol!: 24
Enter int values for ur array lol!: -10
Enter int values for ur array lol!: 9
Enter int values for ur array lol!: -71
Enter int values for ur array lol!: 42
x[0][0] = 12
x[0][1] = 31
x[0][2] = 45
x[1][0] = -20
x[1][1] = 24
x[1][2] = -10
x[2][0] = 9
x[2][1] = -71
x[2][2] = 42

```

-----oOo-----

Further reading and digging:

1. [Check the best selling C/C++ books at Amazon.com.](#)