

Programming Language-II

Lecture # 8

Lecture Content

- What is an array?
- Advantages and limitations
- Types of array
- Single dimensional arrays(1D)
 - Its declaration
 - Its definition
 - Passing it to functions
 - Examples
 - Its Manipulation (Searching, Sorting, Insert element, Delete element)
 - char type array(strings)
- Multi dimensional arrays(2D)
 - Its declaration
 - Its definition
 - Examples
 - Passing it to functions

What Is An Array?

- An array is collection of items stored at continuous memory locations. The idea is to declare multiple items of same type together.
- An array is a collection of values that have the same data type, e.g.
 - A collection of int data values or
 - A collection of bool data values
- We refer to all stored values in an array by its name

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

Array Length = 9

First Index = 0

Last Index = 8

Advantage of array

- It is used to represent multiple data items of same type by using only single name.
- It can be used to implement other data structures like linked lists, stacks, queues, trees, graphs etc.
- 2D arrays are used to represent matrices.

Limitations of array

- We must know in advance that how many elements are to be stored in array.
- Array is static structure. It means that array is of fixed size. The memory which is allocated to array can not be increased or reduced.
- The elements of array are stored in consecutive memory locations. So may be consecutive location is not available in memory so in this scenario program will create exception.
- The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.

Types of an array

- Single dimensional(1D)
 - `int arr[5];`
- Multi dimensional(2D)
 - `int arr[5][5];`

Ways to Declaring 1D arrays

- Array declaration by specifying size

```
int arr[10];
```

- Array declaration by initializing elements

```
int arr[] = {10, 20, 30, 40}
```

Compiler creates an array of size 4 and above is same as "int arr[4] = {10, 20, 30, 40}"

- Array declaration by specifying size and initializing elements

```
int arr[6] = {10, 20, 30, 40}
```

Compiler creates an array of size 6, initializes first 4 elements as specified by user and rest two elements as 0 and above is same as "int arr[] = {10, 20, 30, 40, 0, 0}"

Accessing Array Elements

Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1. Following are few examples.

```
int main()
{
    int arr[5];
    arr[0] = 5;
    arr[1] = -10;
    arr[2] = 2;
    arr[3] = arr[0];

    cout<<arr[0]<<endl
         << arr[1] <<endl
         << arr[2] <<endl
         << arr[3] <<endl;

    return 0;
}
```


One-dimensional Arrays

- We use one-dimensional (1D) arrays to store and access list of data values in an easy way by giving these values a common name, e.g.

int x[4]; // all values are named x

x[0] = 10; // the 1st value is 10

x[1] = 5; // the 2nd value is 5

x[2] = 20; // the 3rd value is 20

x[3] = 30; // the 4th value is 30

10	x[0]
5	x[1]
20	x[2]
30	x[3]

Array Out-of-bound Run-time Error

- All C++ one-dimensional arrays with N entries start at index 0 and ended at index N-1
- `const int N=5;`
- `int v[N]; // this array contains 5 entries`
- It is a common error to try to access the Nth entry, e.g. `v[5]` or `V[N]`, since the index of the last array entry is N-1, not N

Initializing One-dimensional Arrays

- There are two common ways to initialize one-dimensional arrays

- Using for loop, e.g.

```
int x[10];  
for( int index=0; index<10; index++)  
    x[index] = index+1 ;  
for (int index=0; index<10; index++)  
    cin >> x[index];
```

- Specifying list of values while declaring the 1D array, e.g.

```
int x[10] = {1,2,3,4,5,6,7,8,9,10};  
int y[ ] = {0,0,0}; // this array contains 3 entries with 0 values  
double z[100] = {0}; // this array contains 100  
    // entries, all of which are initialized by 0  
double w[20] = {5,3,1}; // this array contains 20 entries,  
    // the first three entries are initialized by 5, 3, and 1 respectively  
    // while the remaining 17 entries are automatically initialized by 0  
bool pass[10] = {true, true}; // this array contains 10 entries.  
    // The first two entries are initialized to true, while the remaining 8  
    // entries are automatically initialized to false
```

Displaying Values Stored in 1D Arrays

Wrong way:

```
Int x[10];  
cout << x;
```

Correct way:

```
int x[10];  
// Displaying one value per line to the user:  
for (int index=0; index<10; index++)  
    cout << x[index] << endl;
```


Example: Read Values and Print them in Reverse Order

```
void main()  
{
```



```
C:\C++Projects\Test1\Debug\Test1.exe  
Please enter five integer values: 10 20 30 40 50  
The values in reverse order are: 50 40 30 20 10
```

```
int x[5], index;  
cout << "Please enter five integer values: ";  
for( index=0; index<5; index++) cin >> x[index];  
cout << "The values in reverse order are: ";  
for (index=4; index>=0; index--)  
    cout << x[index]<<endl;
```

```
}
```

Passing 1D Arrays to Functions

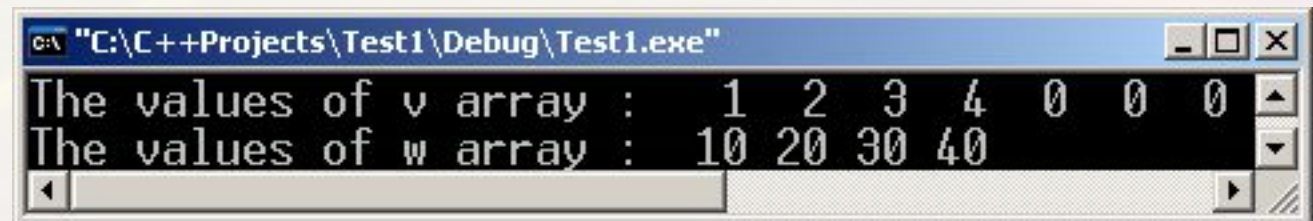
- By default, arrays are passed to functions by reference even though we do not even write the reference operator (&) in front of the array name, why?
 - Since arrays store huge number of values, it is much faster and more economical to pass arrays by reference instead of wasting the time and memory copying the values of the arrays into another arrays
- Example
 - The following function signature accepts an array x as a reference parameter and it also accepts the number of entries in the array (i.e. its size)
void process (int x[], int size);

Example: Passing 1D Array to A Function

```
#include <iostream>
#include <string>
using namespace std;

void display ( int x[ ], int n)
{
    for (int i=0; i<n; i++)
        cout << x[i];
    cout << endl;
}

void main()
{
    int v[7] = {1,2,3,4};
    int w[4] = {10,20,30,40};
    display (v, 7);
    display (w, 4);
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\C++Projects\Test1\Debug\Test1.exe". The window contains the following output:

```
The values of v array : 1 2 3 4 0 0 0
The values of w array : 10 20 30 40
```

The output demonstrates that the `display` function correctly prints the elements of the `v` array (1, 2, 3, 4) and the `w` array (10, 20, 30, 40). The `v` array is printed with 7 elements, including the trailing zeros, while the `w` array is printed with 4 elements.

Passing 1D Arrays As const Reference Parameters

- It is a good idea to pass arrays as const reference parameters if you want to assure that the functions will never change the values stored in the arrays, e.g.

```
void display (const int x[ ], int size)  
{  
    for (int i=0; i<size; i++)  
        cout << x[i] << endl;  
}
```


Exercise

- 1) Take 10 integer inputs from user and store them in an array and print them on screen.
- 2) Write function which will return sum of all element from an array.
- 3) Write function which will return smallest element from an array.
- 4) Write function which will return largest element from an array.

Exercise

- 5) Take 20 integer inputs from user and print the following:
number of positive numbers
number of negative numbers
number of odd numbers
number of even numbers
number of 0.
- 6) Take 10 integer inputs from user and store them in an array. Again ask user to give a number. Now, tell user whether that number is present in array or not.

Exercise

- 7) Take 10 integer inputs from user and store them in an array. Now, copy all the elements in another array but in reverse order.
- 8) Write a program to check if elements of an array are same or not if read from front or back. E.g.-

2	3	15	15	3	2
---	---	----	----	---	---

- 9) Write function which takes two arrays as input and finds whether 1st array is sub array of 2nd array or not.

Example: Searching for the smallest value in an array

```
int smallest (const int x[ ], int n)
{
    /* Assume for a moment that x[0] contains
       the smallest value */
    int min = x[0];
    for (int i=1; i<n; i++)
        if (min>x[i])
            min = x[i];
    return min;
}
```


Example: Searching for A Key in Unsorted Array Using Sequential Search

```
int seqSearch (int x[ ], int n, int key)
{
    for (int i=0; i<n; i++)
        if (key == x[i]) return i;
    return -1;
}
```

This function will return the index of the first value that equals to the key if exists, otherwise, it returns -1

Searching for The Index of The Smallest Value in An Array

```
int min_index (int x[ ], int size, int start=0)
{
    int index = start;
    int min = x[index];
    for (int i=index+1; i<size;
        if (min>x[i])
        {
            min = x[i];
            index = i;
        }
    return index;
}
```

Notice that:

1. The array is not sorted!
2. We use the parameter start to specify the start searching index. The 0 is the default value of this parameter
3. Later on, we will use this function to sort an array

Sorting 1D array

```
int main(){
    int data[] = { 7, 3, 9, 2, 5 };
    int tmp;

    for (int i = 0; i < 5; i++)
        cout << data[i] << " ";
    cout << endl;
    for (int i = 0; i < 5 - 1; i++)
        for (int j = i + 1; j < 5; j++)
            if (data[i] > data[j])
            {
                tmp = data[i];
                data[i] = data[j];
                data[j] = tmp;
            }
}
```

Sorting 1D array(Cont...)

```
for (int i = 0; i < 5; i++)  
    cout << data[i] << " ";  
  
cout << endl;  
system("pause");  
return 0;  
}
```


Insertion in 1D array

```
int main(){
int a[20], n, x, i, pos = 0;
cout << "Enter size of array:";cin >> n;
cout << "Enter the array in ascending
order:\n";
for (i = 0; i<n; ++i)
    cin >> a[i];
cout << "\nEnter element to insert:";
cin >> x;
for (i = 0; i<n; ++i)
    if (a[i] <= x&& x<a[i + 1]){
        pos = i + 1;
        break;}
}
```

Insertion in 1D array

```
for (i = n; i>pos; --i)
a[i] = a[i - 1];

a[pos] = x;
n++;
cout << "\n\nArray after inserting element:\n";

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

Deletion from 1D array

```
int main(){
int arr[50], size, i, del, count = 0;
cout << "Enter array size : ";
cin >> size;
cout << "Enter array elements : ";
for (i = 0; i<size; i++){
cin >> arr[i];
}
cout << "Enter element to be delete : ";
cin >> del;
for (i = 0; i<size; i++){
    if (arr[i] == del){
        for (int j = i; j<(size - 1); j++){
            arr[j] = arr[j + 1];
        }
        count++;break;}}}
```

Deletion from 1D array(Cont...)

```
if (count == 0){
    cout << "Element not found..!!";
}
else{
    cout << "Element deleted successfully..!!\n";
    cout << "Now the new array is :\n";
    for (i = 0; i < (size - 1); i++){
        cout << arr[i] << " ";
    }
}
system("pause");
return 0;}
```


Strings in C

Strings

- A string is an Null-Terminated Character Array
- Allocate space for a string just like any other array:
`char outputString[16];`
- Space for string must contain room for terminating zero Special syntax for initializing a string:

```
char outputString[] = "Result = ";
```

...which is the same as:

```
outputString[0] = 'R';
```

```
outputString[1] = 'e';
```

```
outputString[2] = 's';
```

```
...
```

```
outputString[9] = '\0'; // Null terminator
```

String Declaration & initialization ways

Declaration plus initialization at same time

```
char name[ ] = "Idrees Ahmad";  
cout << name << endl;
```

Initialize from user:

```
char name[20];  
cin >> name;  
cout << name << endl;
```

Note: But in this way compiler only read characters up to first space and ignore all remaining's.

getline function

- To read string included space we can use getline function,

```
char name[20];  
cin.getline(name, 20);  
cout << name << endl;
```


Exercise

- 1) Write code to count no of character in string.
- 2) Write code to count no of spaces is present in a string.
- 3) Take two string from user and check either both are same or not.
- 4) Write code to copy first string to another.

Two dimensional array

- `type arrayName [x][y];`

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Initializing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {  
    {0, 1, 2, 3} , /* initializers for row indexed by 0 */  
    {4, 5, 6, 7} , /* initializers for row indexed by 1 */  
    {8, 9, 10, 11} /* initializers for row indexed by 2 */  
};
```

OR

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

```
int main() {  
    // an array with 5 rows and 2 columns.  
    int a[5][2] =  
    { { 0,0 }, { 1,2 }, { 2,4 }, { 3,6 }, { 4,8 } };  
  
    // output each array element's value  
    for (int i = 0; i < 5; i++)  
        for (int j = 0; j < 2; j++)  
        {  
            cout << "a[" << i << "][" << j << "]: ";  
            cout << a[i][j] << endl;  
        }  
    return 0;  
}
```


Passing 2D array to function(When dimensions are globally define)

```
const int R = 3;
const int C = 3;

void print(int arr[][C]);

int main(){
int arr[][C] = {
    { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }
};
print(arr);
system("pause");
return 0;
}
```

Passing 2D array to function(When dimensions are globally define)(Cont...)

```
void print(int arr[R][C])
{

    int i, j;
    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            cout<< arr[i][j]<<endl;
}
```

Passing 2D array to function(When only second dimension is available globally)

```
const int C = 3;
```

```
void print(int arr[][C], int);
```

```
int main()
```

```
{
```

```
    int arr[][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
    print(arr, 3);
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

Passing 2D array to function(When only second dimension is available globally)(Cont...)

```
void print(int arr[][C], int R)
{
    int i, j;
    for (i = 0; i < R; i++)
        for (j = 0; j < C; j++)
            cout<< arr[i][j]<<endl;
}
```


Example

3	5	7	-1	14
4	8	5	1	18
0	3	7	2	12
6	7	7	1	21

Home work

- Write code to take two matrixes from user having dimensions **5x5** and implement given below functions,
 - Matrix addition
 - Matrix subtraction
 - Matrix multiplication

Matrix multiplication

```
void matrixMul(int matrix1[][C], int matrix2[][C], int
resultantMatrix[][C])
{
    int sum;
    for (int i = 0; i<3; i++)
    {
        for (int j = 0; j<3; j++)
        {
            sum = 0;
            for (int k = 0; k<3; k++)
            {
                sum = sum + matrix1[i][k] *
matrix2[k][j];
            }
            resultantMatrix[i][j] = sum;
        }
    }
}
```