



**University of Engineering and Technology,  
Peshawar, Pakistan**

---

# CSE 102: Computer Programming

## Lecture 04 Arrays

By;  
Dr. Muhammad Athar Javed Sethi

# Array

- An array is a set of variables, represented by a single name.
- The individual variables are called elements and are identified by index numbers.
- The following example declares an array with ten elements.

```
type array_name[n];
```

- e.g.

```
int myarray[10];
```

# Indexing

- The first element in the array has an index number of zero.
- Therefore, the array `myarray` has ten elements, indexed from zero to nine.

`myarray[0]`

`myarray[1]`

▪

▪

`myarray[9]`

# Accessing the Elements

- To access an individual element in the array, the index number follows the variable name in square brackets.
- The variable can then be treated like any other variable in C++.
- The following example assigns a value 16 to the first element in the array i.e.

```
myarray[0] = 16;
```

# Initializing Array Elements

- Arrays can be initialized like any other variables by assignment.
- As an array contains more than one value, the individual values are placed in curly braces, and separated with commas.
- The example initializes a ten dimensional array with the first ten values of the three times table.

```
int x[10] = {3, 6, 9, 12, 15, 18, 21, 24, 27, 30};
```

# Looping through an Array

---

- As the array is indexed sequentially, we can use the for loop to display all the values of an array.
- The example displays all the values of an array.

# Example

```
#include<iostream.h>
```

```
Using namespace std;
```

```
int main ()
```

```
{
```

```
    int x[10]={3,6,9,12,15,18,21,24,27,30};
```

```
    int i;
```

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

```
        cout<< "x["<<i<<"]= "<< x[i]<<endl;
```

```
    return 0;
```

```
}
```

# Assignment

- Assigning the values individually

```
int x[10];
```

```
x[0] = 3;
```

```
x[1] = 6;
```

```
x[2] = 9;
```

```
x[3] = 12;
```

```
x[4] = 15;
```

```
x[5] = 18;
```

```
x[6] = 21;
```

```
x[7] = 24;
```

```
x[8] = 27;
```

```
x[9] = 30;
```



# Working with Arrays

- Example: adds up the elements of an array:

```
int array[5]={2, 3, 4, 5,2}  
int sum = 0;  
for(int i=0; i<5; i++)  
    sum = sum + array[i];
```

# Arrays of Different Types

```
int ID[30];
```

```
/* Could be used to store  
the ID numbers of students in a class */
```

```
float temperatures[31];
```

```
/* Could be used to store the  
daily temperatures in a month */
```

```
char name[20];
```

```
/* Could be used to store a  
character string */
```

# String Variable

- String variable is an array of character type.
- Length of the string is the total number of elements of the array.
- Syntax is;

`Char variable_name[n];`

- Example
  - `name[15];`
  - `city[10];`
- Last character of every string variable is null character.
- Null character is represented by ‘\0’
- If null character is not present at the end of string then the variable of char type is handled as an array not as string.

# string Functions

- “string” header file contains the functions that are used to process strings
- `stringVariable. size();`
- `getline (cin, stringVariable);`  
`string3.assign( string1 );`
- `first.swap( second );`
- For other string functions, see book.

# Sorting Arrays

- Process of arranging data in a specified order is call sorting.
- Numeric type data can be arranged in ascending or descending order.
- Character type data may be arranged in alphabetical order.
- Methods
  - Bubble Sort (Assignment)
  - Selection Sort (Assignment)

# Multidimensional Arrays

- An array can have more than one dimension.
- By allowing the array to have more than one dimension provides greater flexibility.
- Spreadsheets are built on a two dimensional array; an array for the rows, and an array for the columns.
- Syntax;  
    `type array_name[r] [c];`

# Initializing Tables (Arrays)

- For Example;

- `Int myarray[2][3] = { {50,66,82} ,  
{19,92,106}};`

- `Char myarray[5][3] = { {'a', 'b' , 'c'},  
{'d', 'e' , 'f'},  
{'1', '2' , '3'},  
{'f', 'g' , 'h'},  
{'4', '5' , '6'}};`

# Example

---

- The example uses a two dimensional array with two rows, each containing five columns.



```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    /* Declare a 2 x 5 multidimensional array */
```

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

```
    int row, column;
```

```
    for (row=0; row<2; row++)
```

```
    {
```

```
        for (column=0; column<5; column++)
```

```
            cout<< x[row][column]<<endl;
```

```
    }
```

```
}
```

# Storage in Memory

---

- Since computer memory is essentially one-dimensional, with memory locations running straight from 0 up through the highest location in memory.
- A multidimensional array cannot be stored in memory as a grid. Instead, the array is dissected and stored in rows.
- Consider the two-dimensional array.

# Storage

```
-----  
row 0 | 1 | 2 | 3 |  
-----  
row 1 | 4 | 5 | 6 |  
-----  
row 2 | 7 | 8 | 9 |  
-----
```

- To the computer, the array above actually "looks" like this:

```
-----  
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  
-----  
| row 0 | row 1 | row 2 |
```