Fundamentals of Computer Programming

Chapter 4
Array and Strings

Chere L. (M.Tech)
Lecturer, SWEG, AASTU

Outline

- Basic concepts of Array
- Types of Array
  - ✓ *One Dimensional Arrays*
  - ✓ *Multi-dimensional Arrays*
- Array declaration and initialization
- Accessing and processing Array Elements
- Basics of String
- String declaration and initialization
- String manipulation and operation
  - ✓Input/output, Copying, Comparing, concatenation, etc.
- String library functions and operators

Part I
Array

What is an array?

- An array is a group of **consecutive memory locations** with **same name and data type**.
- So far we were dealing with **scalar and atomic variable** is a single memory location with unique name and a type.
- Unlike **scalar and atomic variable** an **array** is collection of different adjacent memory locations.
- Can be also referred as
  - Series of elements (variables) of the same type placed consecutively in memory.
  - Memory collection to hold more than one value of the same types at a time (it's like of list of items).
- Array is derived data structure which built up on primitive data types

1. Basic concepts of Array (Cont'd)

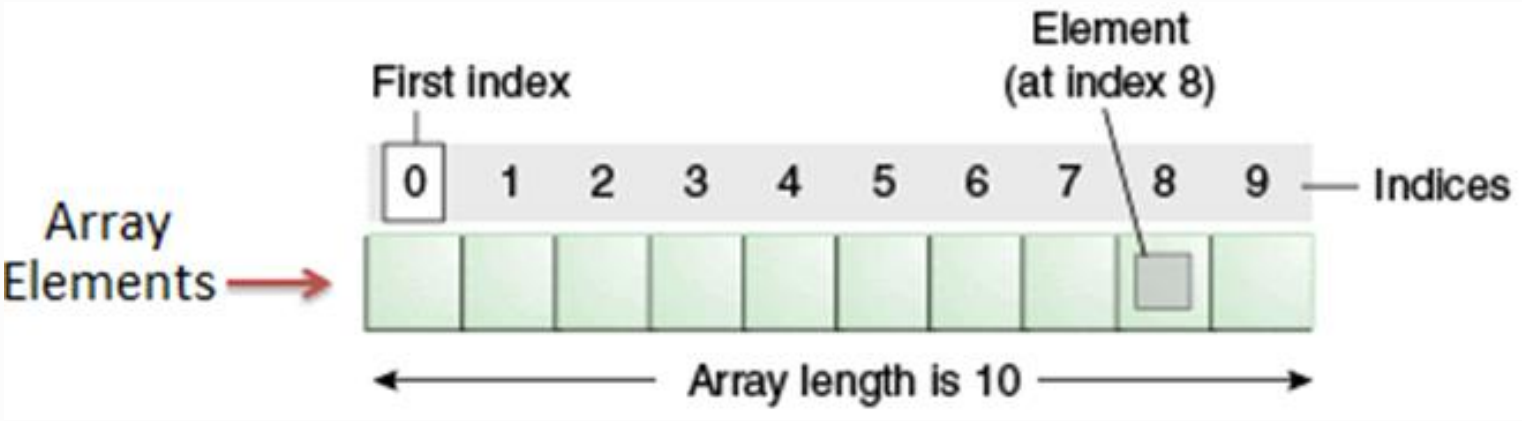
In general array concept comprises the following

- Array is a consecutive group of memory locations
- All these memory locations referred by a collective name and have the same data type (either primitive/non primitive).
- Elements of array
  - the memory locations in the array/value stored.
- Length of array
  - define the total number of elements in the array
  - Must be *constant natural number* (most probably >=2)
- The elements of array is accessed with reference to its position in array, that is call index or subscript.
- Both the length of array and index/subscript of an array are specified within **square bracket []**

1. Basic concepts of Array (Cont'd)

- Array index range – starts from 0 and extend till the size of the array minus one.

  i.e. [0 → length – 1]



Need of array - *Consider the scenario given below…*

- Write a program that reads five numbers, and performs some manipulations on these numbers, such as find their sum, and print the numbers in reverse order, find the their mean and compute standard deviation etc.
- **Solution:**
  - ✓ We could use five individual variables of type **int,** but five variables are hard to keep track of.
  - ✓ We could make program more readable by giving the variables related names such as *item1, item2, item3*, and so forth (see next slide)
  - ✓ However, this solution becomes absurd/challenging if the number of **items is very large** *(if we will have hundred list of numbers)*
- **Therefore**, the use of **array** is the best solution for such scenario

1. Basic concepts of Array (Cont'd)

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

int main()
{
    int item0, item1, item2, item3, item4;
    int sum;

    cout<<"Enter five integers separted by space: ";
    cin>>item0>>item1>>item2>>item3>>item4;

    sum = item0 + item1 + item2 + item3 + item4;

    cout<<"\nThe sum of the numbers = "<<sum<<endl;
    cout<<"The numbers in reverse order are: ";
    cout<<item4<<" "<<item3<<" "<<item2<<" " << item1<<" "<<item0<<endl;
    return 0;
}
```

What if the size of the given number is too large?

- Instead of declaring different variables each with a different identifier, using an *array* we can store any size of different values of the same type.

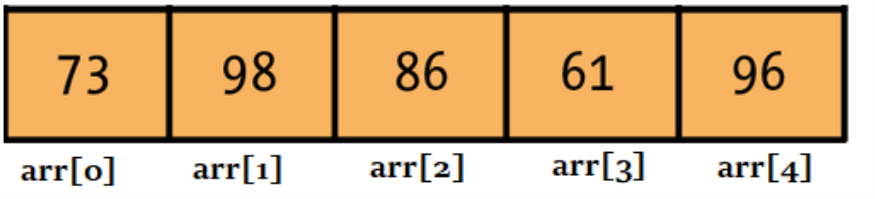
Advantages/Use array

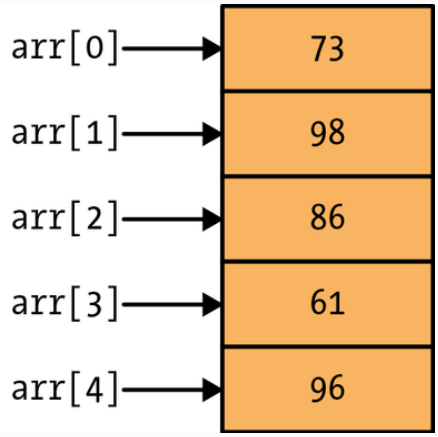
- Arrays can store a large number of value with single name.
- Arrays elements are easily referred after certain operation carried out because array maintain the original value unless we change it.
- Arrays are used to process many value easily and quickly.
- Array mainly applicable in large data set sort and search operations
  - ✓ The values stored in an array can be sorted easily.
  - ✓ The search process can be applied on arrays easily

2. Types of Array

- Mainly there are two types of array namely **One dimensional** and **Multi dimensional**

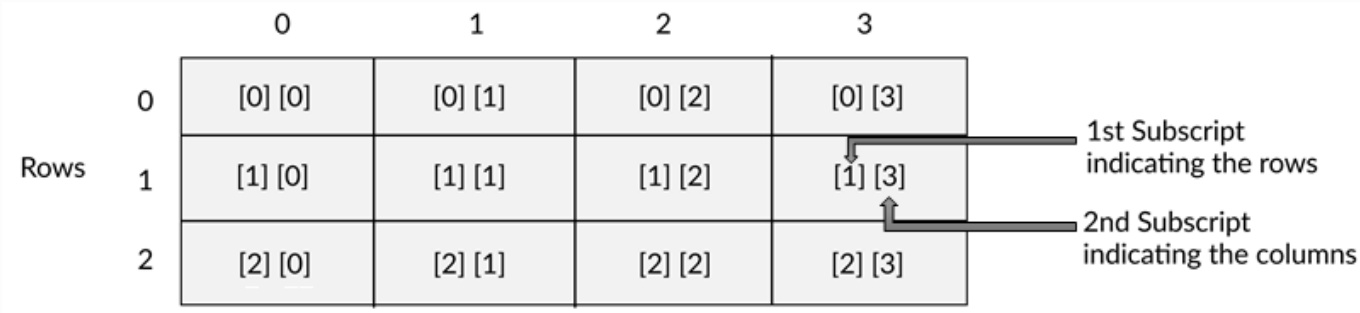
- **One Dimensional Array**
  - ✓ Simplest form of array which has only single subscript/index
  - ✓ Also called vectors
  - ✓ Represented either horizontally (single row) or vertically (single column) as described below





2. Types of Array (cont'd)

- **Two Dimensional Array**
  - ✓ Simplest form of Multi-Dimensional array
  - ✓ It is a row and column based data structure
  - ✓ Referred to as a matrix or table
  - ✓ A matrix has two subscripts/indices, one denote the row and another denotes the column.
  - ✓ In other words, 2D array is an array of 1D arrays



- **Multi Dimensional Array**
  - ✓ An array with more than two dimension, like 3D, 4D etc.

3. Array Declaration and Initialization

- **Declaration**
  - ✓ The process of reserving memory location for an array elements
  - ✓ Refers to **Naming the array**, specifies the **type of its elements** and also define the **number of elements** in the array.
  - ✓ Like any other scalar variable, an array must be declared before it is used.

- **Initialization -** assigning initial value to the **elements of** an array
  - If we declare a **global array** its content will be initialized with all its elements filled **with zeros by default**.
  - However, when declaring an **array of local scope** and if we do not specify the initial values, it will not be initialized and its content is undetermined until we store some values in it.

3. 1) Declaration of 1D Array

- **Syntax:** **data_type identifier[length];**

  ✓ Data _type: Data type of values to be stored in the array.
  e.g. int, double, float, long etc.

  ✓ Identifier: Name of the array
  governed by the same rules of variable naming.

  ✓ Length: Number of array elements
  - Should be constant natural number
  - Specified inside square brace.
  - **array size = size of dataType * length**

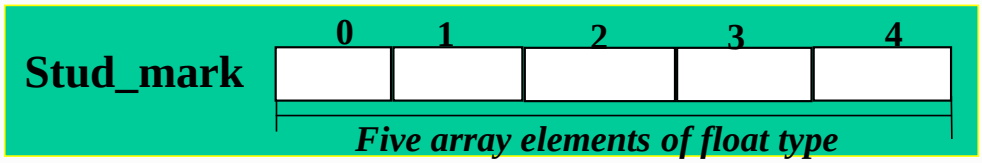
- **Example 1:**

```
float stud_Mark [30];
int itemQuantity [13];
```

3. 1) Declaration of 1D Array (Cont'd)

- **Example 2:** defining an array size as **constant**

```
const int arraySize = 5;
float stud_Mark [arraySize];
```



- **Example 3:** declaring multiple array of the same type

```
float stud_Mark [20],   item_Price[15],  emp_Salary[33];
```

NOTE:

- *The elements field within brackets [] when declaring an array must be a **constant value,** since arrays are blocks of static memory of a given size and the compiler must be able to determine exactly how much memory it must assign to the array before any instruction is considered.*

3.2) Initialization of 1D Array

- **Alternative 1: initializing array elements during array declaration**
  - ✓ **Syntax:**

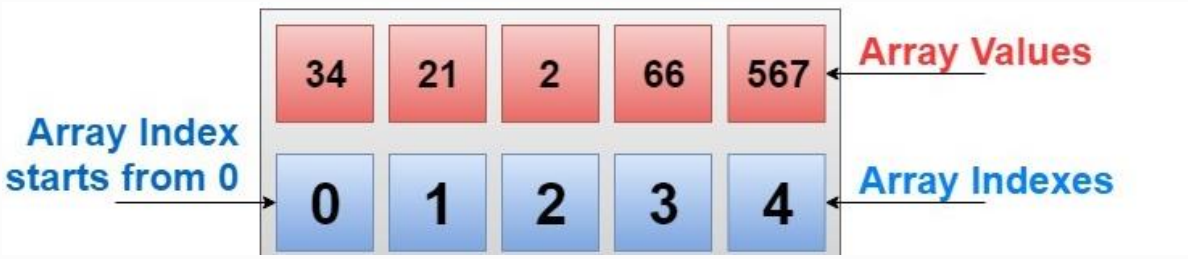
    data_type identifier[length] = {V1, V2, …… Vn};

    where n <= length and

    V1, V2, . . Vn is list of initial values

  - ✓ **Example 1:**

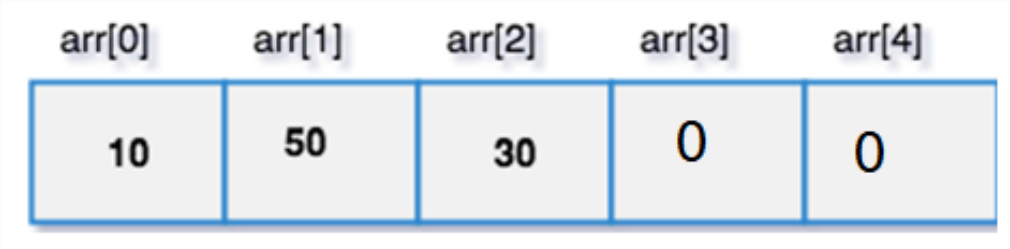
    int marks[5] = {34, 21, 2, 66, 567};



3.2) Initialization of 1D Array

✓ **Example 2:**

```
float arr[5] =  {10,  50, 30};          //less initial value
```



✓ **Example 3:**

```
float salary[3] = {0};      //initialize all array
float salary[3] = {};       //elements to zero

float salary[3]={57, 154,  82,  96,  40};   //invalid,
                                            //out of bound
```

3.2) Initialization of 1D Array

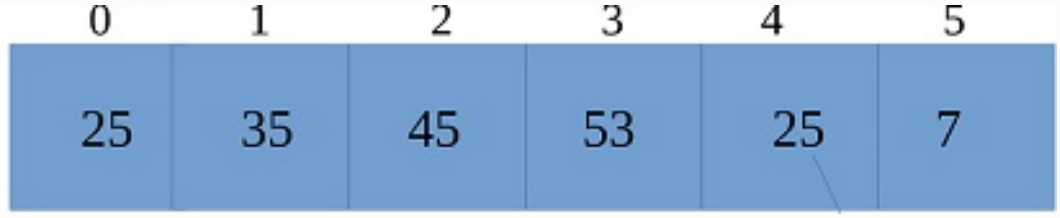
- **Alternative 2: omitting the size while initializing an array elements**
  - ✓ **Syntax:**

    data_type identifier[ ] = {V1, V2, …… Vn};

    - Here the size of array is determined by the size of initial value of an array
    - i.e. length = size of initial value
    - Such array is also called **unsized array**

  - ✓ **Example:**

```
float salary[ ]={25, 35,  45,  53,  25, 7};   //array length = 6
```



3.2) Initialization of 1D Array (Cont'd)

- **Alternative 3: initializing after array declaration**
  - ✓ After declaration array elements can be initialized individually by specifying their index.
  - ✓ Most probably it is a kind of run-time initialization

  - ✓ **Example 1: specifying every individual array element**

```
float salary[5 ];
        salary[0] = 570;  salary[1] = 1054;
        salary[4] =  862;
        salary[5] = 960; //invalid, array out of bound
```

  - ✓ **Example 2: using loop when the initial array elements are consecutive**

```
int myArray [ 20];
for(int i=0; i<20; i++){
        myArray[i] = 10 + (i*i) }
```

3. 3) Declaration of 2D Array

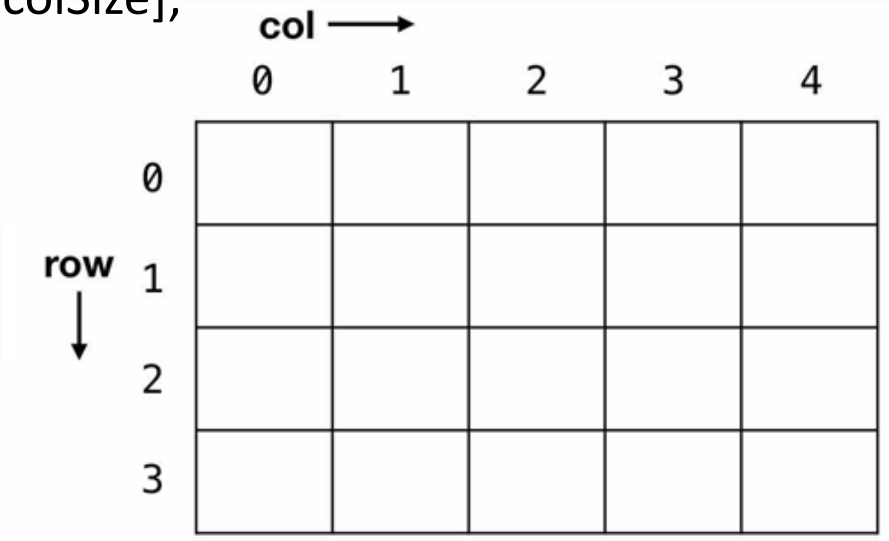
- **Syntax:** **data_type identifier[rowSize][columnSize];**
  - ✓ Data _type: Data type of values to be stored in the array.
    e.g. int, double, float, long etc.
  - ✓ Identifier: Name of the array
    governed by the same rules of variable naming.
  - ✓ rowSize & columnSize : specify row and column size of an array
    - Should be constant natural number
    - Both of them should be specified inside square brace
    - **array size = size of base dataType * rowSize * colSize**
- **Example 1:**
  - declaring 2D array that store mark of 30 students for 6 subjects

    float stud_Mark [30][6];

3. 3) Declaration of 2D Array (Cont'd)

- **Example 2:**

```
const int rowSize = 4, colSize = 5;
float stud_Mark [rowSize ][colSize];
```



NOTE:

- *The row size and column size specified with **brackets []** when declaring an array must be a **constant value,** since arrays are blocks of static memory of a given size and the compiler must be able to determine exactly how much memory it must assign to the array before any instruction is considered*

3.4) Initialization of 2D Array

- **Alternative 1: initializing array elements during array declaration**
  - ✓ **Syntax (option 1): just list array elements within curly brace**

    - **data_type identifier[rowSize][colSize] = {V1, V2, V3, … Vn};**

      where *n <= array Length*, and *i, k refers to row indices*

    - **Example 1:** int bonus[2][3] = { 1, 2, 3, 4, 5, 6};

  - ✓ **Syntax (option 2): grouping array elements by row**

    - **data_type identifier[rSize][cSize] = { {Vi1, Vi2, .. Vin},**
      {Vk1, Vk2, . . .Vkn },
      { . . . }};

    - **Example 2:**

      int bonus[4][3] = { {1, 2, 3}, {4, 8, 12}, {7, 8, 9}, {0}};

3.4) Initialization of 2D Array (Cont'd)

✓ **Example 3:** partial initialization (leave some array elements uninitialized

```
float price[4][3] = {  {1.5, 2.3, 3.5},
                       {4.5, 6.5}, {7.5},
                       {10.7, 30} };
```

Array elements can span multiple lines

✓ **Example 4:** initializing all array elements to zero

```
int bonus[4][3] = {0};   or

int bonus[4][3] = {};
```

NOTE:

- Grouping elements are advantages to make partial initialization.

3.4) Initialization of 2D Array (Cont'd)

- **Alternative 2: omitting array size while initializing array elements**
  - ✓ **Syntax :** **data_type identifier[][colSize] = {V1, V2, … Vn};**
    - also referred as **Unsized** array
  - ✓ **Example:**

```
int theArray[ ][3] ={ 1, 2, 3, 4, 5,
                      6, 7, 8, 9, 10, 11,
                      12, 13, 14, 15, 16};
int myArray[ ][3] ={ 4, 5, 6, 7, 8, 9, 10, 11, 12};   // rSize = 9/3 = 3
int stud_Array[ ][3] = { {1,2}, {3,4}, {5} } ;     // rSize = #group = 3
int item[][] = { {1,2}, {3,4} } ;     // invalid.
```

rowSize = 16/3 = 5.11 = 6

NOTE:

- *In all multidimensional array it must have bounds for all dimensions except the first.*

3.4) Initialization of 2D Array (Cont'd)

- **Alternative 3: initializing after array declaration**
  - ✓ After declaration array elements can be initialized individually by specifying their index.
  - ✓ Most probably it is a kind of run-time initialization

- **Example 1:** `float item[3][2];  //declaration`

```
item[0][0] = 10;    item[1][0] = 17;    item[2][0] = 13;
item[0][1] = 105;   item[1][1] = 120;   item[2][1] = 102;
```

- **Example 2 (rare case):** `float invoice[3][5];  //declaration`

```
for (int i = 0; i < 3; i++)
  for (int j=0; j < 2; j++)
      invoice [i][j] = 2*(i+j);
```

Multi-Dimensional (3D Array)

- **Syntax:**

  DataType arrayName [x][y][z];

- **Example:**

  ✓Design a program that read and stores the mark of SWEG 2012 batch of all courses they are taking semester wise.

- **Solution**

  ✓ Dimension 1 can be refer to **Student list**, like student ID

  ✓ Dimension 2 can be refer to **academic semester**

  ✓ Dimension 3 can be refer to list of courses the students take in a semester

3.5) Accessing/processing Array Elements

- The values of an array can be access using array index only.

  ***arrayName*[*index*];** **//1D array**

  arrayName [rowIndex][colIndex] **//2D array**

- To store the value of *33* in the 2nd element of 1D array called ***item*** or 2D array called ***value*** see the below:

  ***item*[*1*] = *33*;** **value[0][1] = 33;**

- To pass the value of the second element of ***item*** or ***value*** to the variable *temporary*:

  ***temporary* = *item*[*1*];**

  ***temporary* = value [0][*1*];**

3.5) Accessing Array Elements (cont'd)

(a) Copying array elements:

- C++ does not allow aggregate operations on arrays.
- For example, given array

```
int x[50],   y[50] ;
```

- There is no aggregate assignment of y to x

```
x = y; //not valid
```

- To copy array **y** into array **x**, you must do it yourself, element by element.
- Mostly for loop is used for this purpose

```
for ( i=0; i<50; i++){
        x[i] = y[i];            //valid operation
}
```

3.5) Accessing Array Elements (cont'd)

(b) Comparing two array elements:

- Similarly, there is no aggregate comparison of arrays.

```
if (x == y )          //Not valid
```

where **x** & **y** are an array declared in previous example

(c) Arithmetic operations on two array elements

- Also we cannot perform aggregate arithmetic operations on arrays also.

```
x = x + y     // not valid, where x and y are an array.
```

- Moreover, it is not possible to return an entire array as the value of a value-returning function

```
return x;    //not valid, where x is an array.
```

3.5) Accessing Array Elements (cont'd)

(d) Input/output array elements:

- Consider an array declaration

```
float arr[30];
```

- Also, aggregate input / output operations are not supported on arrays in C++.

```
cin>>x;      //not valid, where x is an array
cout<<x;     //not valid, but it prints address of the array
```

- Mostly we use loop to input/output array elements
- Example:

```
for ( i=0; i<30; i++)
cin>>arr[i];
```

```
for ( i=0; i<30; i++)
cout<<arr[i];
```

//valid operation

3.5) Accessing Array Elements (cont'd)

Note:

- In C++ it is perfectly valid to exceed the **range of indices** for an Array, since it does not cause compilation errors.
- However, it is problematic at runtime which can cause unexpected results or serious errors during execution.

- It is very important to clearly distinguish between the two uses that brackets [ ] have related to arrays
  - ✓ To set the size of arrays when declaring them
  - ✓ To specify indices for a concrete array element when referring to it

Example program 1

- **A Program to demonstrate 1D array Declaration, initialization, manipulation and processing**

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

int main() {
   int const SIZE = 5;

    int a1[SIZE];   // Uninitialized
    cout<<"Uninitialized elements of array: "<<endl;
    for (int i = 0; i < SIZE; ++i)
        cout << a1[i] << " ";
    cout <<"\n\n";   // ? ? ? ? ?

    int a2[SIZE] = {21, 22, 23, 24, 25};    // All elements initialized
    cout<<"All array elements are initialized: "<<endl;
    for (int i = 0; i < SIZE; ++i)
        cout << a2[i] << " ";
    cout <<"\n\n";              // 21 22 23 24 25
```

Example program 1 (cont'd)

```
    int a3[] = {31, 32, 33, 34, 35};      // Size deduced from init values
    int a3Size = sizeof(a3)/sizeof(int);
    cout << "Omiting array size and initailize array elements: "<<endl;
    cout << "\tSize of an array is " << a3Size << endl;      // 5
    cout << "\tArray elements are \t";       // 5
    for (int i = 0; i < a3Size; ++i)
        cout << a3[i] << " ";
    cout <<"\n\n";                 // 31 32 33 34 35

    int a4[SIZE] = {41, 42};     //Leading elements initialized, the rests to 0
    cout<<"Partially initialized array elements"<<endl;
    for (int i = 0; i < SIZE; ++i)
        cout << a4[i] << " ";
    cout <<"\n\n";                 // 41 42 0 0 0

    int a5[SIZE] = {0};            // First elements to 0, the rests to 0 too
    cout<<"Initializing all array elements to zero"<<endl;
    for (int i = 0; i < SIZE; ++i)
        cout << a5[i] << " ";
    cout <<"\n\n";                 // 0 0 0 0 0

    int a6[SIZE] = {};             // All elements to 0 too
    cout<<"All array elements are initialized to zero"<<endl;
    for (int i = 0; i < SIZE; ++i)
        cout << a6[i] << " ";
   cout <<"\n\n";                  // 0 0 0 0 0
}
```

Example program 2

- **Program to read five numbers, find their sum, and print the numbers in reverse order**

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

int main()
{
    int item[5];  //declare an array item of five elements
    int sum=0, counter;
    cout<<"Enter five numbers."<<endl;
    for(counter = 0; counter < 5; counter++)
    {
        cin>>item[counter];
        sum = sum + item[counter];
    }
    cout<<"The sum of the numbers is: "<<sum<<endl;
    cout<<"The numbers in reverse order are: ";
    for(counter = 4; counter >= 0; counter--)
        cout<<item[counter]<<" ";
    return 0;
}
```

Example program 3

- **A program to demonstrate Array size declaration as constant**

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

int main()
{
    const int arraySize = 10;

    int myArray[arraySize];
    for(int i=0; i<arraySize; i++)
        myArray[i]=2+2*i;

    cout<<"Element"<<setw(13) <<"Value"<<endl;
    for ( int j = 0; j < 10; j++ )
        cout<<setw(7) << j << setw(13);
        cout<<myArray[j]<<endl;

    return 0;
}
```

Exercises 1

- Design a program that read two matrix and perform the following
  - ✓ Print each input matrix in tabular format
  - ✓ Find the sum of the two matrix and print the resulting matrix in tabular format

  Hint: use two dimensional array

Exercise 1 – Solution

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

#define MAX_ROWS 10
#define MAX_COLS 10

int main()
{
    int mat1[MAX_ROWS][MAX_COLS];
    int mat2[MAX_ROWS][MAX_COLS];
    int res_mat[MAX_ROWS][MAX_COLS];
    int i,j, rows, cols;

    cout<<"Enter 1st Matrix: \n";
    cout<<" Enter no.of rows and columns: ";
    cin>>rows>>cols;

    //user input element of
    //row x col of 1st matrix
    for(i=0;i<rows;i++) {
        for(j=0;j<cols;j++){
            cout<<"Enter value for ROW ";
            cout<<i<<" , "<<"COL "<<j<<" : ";
            cin>>mat1[i][j];
        }
    }
```

```
cout<<"Enter 2nd Matrix: \n";
cout<<"Enter no.of rows and columns: ";
cin>>rows>>cols;

//user input element of
//row x col of 2nd matrix
for(i=0;i<rows;i++) {
    for(j=0;j<cols;j++){
        cout<<"Enter value for ROW ";
        cout<<i<<" , "<<"COL "<<j<<" : ";
        cin>>mat2[i][j];
    }
}
```

```
//Display the two matrices
cout<<"Generated table.....\n";
cout<<"/***** Matrix One *****/\n";
for(i=0;i<rows;i++) {
    for(j=0;j<cols;j++) {
        cout<<mat1[i][j]<<" ";
    }
    cout<<"\n";
}
```

Exercise 1 – Solution (cont'd)

```
    //Display the two matrices
    cout<<"Generated table......\n";
    cout<<"/***** Matrix Two *****/\n";
    for(i=0;i<rows;i++) {
        for(j=0;j<cols;j++) {
            cout<<mat2[i][j]<<" ";
        }
        cout<<"\n";
    }

    cout<<"\nThe Result Matrix is: \n";
    for(i=0;i<rows;i++){
        for(j=0;j<cols;j++){
            res_mat[i][j]=mat1[i][j]+mat2[i][j];
            cout<<res_mat[i][j]<<" ";;
        }
        cout<<"\n";
    }
  return 0;
}
```

```
C:\Users\Habesh\Documents\Untitled2.exe
Enter 1st Matrix:
 Enter no.of rows and columns: 2 2
Enter value for ROW 0 , COL 0 : 2
Enter value for ROW 0 , COL 1 : 4
Enter value for ROW 1 , COL 0 : 6
Enter value for ROW 1 , COL 1 : 8
Enter 2nd Matrix:
Enter no.of rows and columns: 2 2
Enter value for ROW 0 , COL 0 : 1
Enter value for ROW 0 , COL 1 : 3
Enter value for ROW 1 , COL 0 : 5
Enter value for ROW 1 , COL 1 : 7
Generated table......
/***** Matrix One *****/
2 4
6 8
Generated table......
/***** Matrix Two *****/
1 3
5 7

The Result Matrix is:
3 7
11 15

--------------------------------
Process exited after 19.45 seconds
Press any key to continue . . .
```

Practical Exercises 1 - Array

1. Write array declarations, including initializers, for the following:
   a) A list of 10 integer voltages: 89, 75, 82, 93, 78, 95, 81, 88, 77, and 82.
   b) A list of 100 double-precision distances; the first six distances are 6.29, 6.95, 7.25, 7.35, 7.40, and 7.42.
   c) A list of 64 double-precision temperatures; the first 10 temperatures are 78.2, 69.6, 68.5, 83.9, 55.4, 67.0, 49.8, 58.3, 62.5, and 71.6.
   d) A list of 15 character codes; the first seven codes are f, j, m, q, t, w, and z.
2) Write a program to declare a 4-by-5 array of integers and initialize with the data 16, 22, 99, 4, 18, -258, 4, 101, 5, 98, 105, 6, 15, 2, 45, 33, 88, 72, 16, and 3.
3) Write a program to input eight integer numbers into an array named temp. As each number is input, add the number into a total. After all numbers are input, display the number and their average.

Reading Resources/Materials

Chapter 11 & 12:

- ✓ Diane Zak; An Introduction to Programming with C++ (8th Edition), 2016 Cengage Learning

Chapter 7:

- ✓ Walter Savitch; Problem Solving With C++ [10th edition, University of California, San Diego, 2018

Link:

- ✓ https://www.w3schools.in/category/cplusplus-tutorial/

Thank You
For Your Attention!!