ARRAYS ……………………………………………………………………..

- One Dimensional Arrays(1D Arrays)
- Processing 1D Arrays
- Searching
- Sorting
- Multidimensional Arrays(2D Arrays)
- Processing 2D Arrays

One Dimensional arrays

Array is a collection of a fixed number of elements all of the same data type and arranged in a list form.

Syntax

number of elements

Type arrayName[arraySize];

type of elements array name

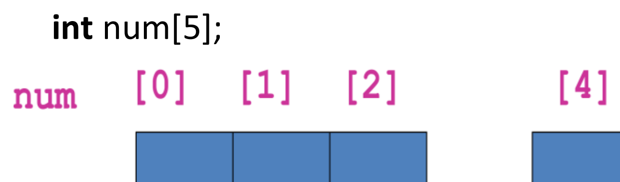
Where:

- **Type** specifies the kind of array elements
- the brackets **[ ]** indicate this is an array
- **arrayName** is the array variable
- **arraySize** is the number of elements

Q: Declares an array **num** of **five** elements. Each element is of type **int**?

Ans:

int num[5];



❑ Accessing Array elements

To access individual elements of an array using:

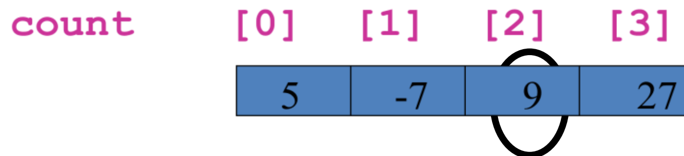
- the name of the array
- a number (index or subscript) that tells which of the element of the array

Syntax:

arrayName[indexExp]

where:

indexExp is any expression whose value is a nonnegative integer within range: between **0** and **arraySize-1**. The index value specifies the position of the element in the array.



What value is stored in **count[2]**?

❑ Declaration and initialization Array

Used when exact size and initial values of an array are known, where:

- We can fully initialize an array in its declaration or partial .
- We can omit the size of an array that is being fully initialized since the size can be deduced from the initialization list.
- **Syntax:**

Type arrayName[arraySize]={initialization list};

or

list of constant expressions of the appropriate *element-type* separated by commas.

Type arrayName[ ]={initialization list};

Examples:

1. **double** sales[5] = {12.25, 32.50, 16.90, 23, 45.68};
2. **int** list[10] = {0};
3. **int** list[10] = {8, 5, 12}; // all other elements to 0.
4. newList = list; // illegal
5. **int** list[5] = {0, 4, 8, 12, 16};

   int newList[5];

   cin >> newList; //illegal

   cout << newList; // illegal

   if (list <= newList) // illegal

❑ Processing Array Elements

Some of the basic operations performed on a one-dimensional array requires the ability to step through the elements of the array. This is easily accomplished using a loop to process each of the elements of the array in turn. For example, suppose that we have the following statements:

```
int list[100]; //list is an array of size 100
for (int i = 0; i < 100; i++)
    // process list[i]
```

- **Steps to read 100 numbers from the keyboard and store the numbers in list:**

```
for (i = 0; i < 100; i++)
  cin >> list[i];
```

- **Steps to printing the elements of array list:**

```
for (i = 0; i < 100; i++)
  cout << list[i] << " ";
```

- **Steps to finding the sum and average of an array list:**

```
sum = 0;
for (index = 0; index < 100; index++)
   sum = sum + list[index];
average = sum / 100;
```

- **Steps to finding Largest element in the array:**

```
largestElem = list[0];
for (index = 1; index < 100; index++)
  if (list[index] >maxValue)
     largestElem =list[index];
```

- **steps to copy one array list into another array newList:**

```
for (int index = 0; index < 5; index ++)
  newList[index] = list[index];
```

Problems:

1. Write program to read N numbers, find their sum, and print the numbers in reverse order.

2. Suppose we need analyze students' test performance in course IS102. We need a program to:

- ❑ let us to enter the test score.
- ❑ compute and display the average.
- ❑ give a report with names, scores, and deviation from the average.

Example

- Display the name in a prompt
- Read the score
- Compute average
- Print summary, including deviation from mean

```
Test Analysis – enter scores:
Fatma    : 92
Sara     : 79
Mohmmed: 95
 . . .
Average = 87.39

Summary …
```

Algorithm:

1. Define **STUDENTS** array to hold names and array **scores** to hold test scores
2. For each student in array
   a) display name & prompt
   b) read double values to store in array scores
3. Compute **average**, display it
4. For each student in array
   a) Display name, test score, difference between that score and **average**

   We need:
   b) array of student names
   c) use of **NUMBER_OF_STUDENTS** constant
   d) for loops to process the arrays

❑ Arrays as Parameters to Functions

In C++ arrays passed as parameters to functions by reference only, the do not use the symbol & when declaring an array as a formal parameter and the size of the array is omitted.

You can pass to function the array's name without an index.

☒ The following program fragment passes the array arr to **func1( )**:

```
int main(void)  {
   int arr[10];
   func1(arr);
   .
   .
   .
 }
```

If a function receives a one-dimension array, you may declare its formal parameter as a sized array, or as an unsized array.

☒ to receive arr, a function called **func1( )** can be declared as:

```
void func1(int x[10]) // sized array
 {
  .
  .
  .
 }
or as:
void func1(int x[]) // unsized array
{
  .
  .
  .
}
```

note:

- C++ does not allow functions to return a value of the type array.

☒ **typedef statement**

typedef statement use for defines a new data type.

Syntax:

typedef type typeName;

Example: To declare array with 20 element for store scores for 20 student, there two forms:

Form1:

```
const int NO_OF_STUDENTS = 20;
int testScores[NO_OF_STUDENTS];
```

Form 2:

```
const int SIZE = 50;
typedef double list[SIZE];
list yourList, mylist;
```

```
double yourList[50];
double myList[50];
```

❑ Sorting an array

Arranging items in a list ascending or descending order is one of the most common operations performed on a list. There are several algorithms to accomplish this – some are known as:

- bubble sort
- selection sort
- quicksort

Bubble sort: This algorithm is efficient for small lists the simplest sorting algorithm.

```
void  bubbleSort( int list[], int listLength) {
   int flag=1;
   while (flag==1 )
    {  flag =0;
      for (int i=0 ; i<listLength-1; i++)
        if (list[i] >list[i+1])
         {
            swap(list[i], list[i+1];
            flag=1;
          }// if
     }//while
}// bubbleSort
```

❑ Searching in array

Searching a list for a given item is one of the most common operations
performed on a list. There are several algorithms to accomplish this – some are known as:

- Sequential search
- Binary search

The following function for the simplest search algorithm called the **sequential search** or **linear search**, it tasks include:

- begin with first item in a list
- search sequentially until desired item found or reach end of list
- with n items in the list, may require n comparisons to find target

```
int seqSearch( int list[], int listLen, int searchItem) {
     int loc;
     loc = 0;
     while (loc < listLen && list[loc] != searchItem)
           loc++;
     if (loc<listLen)          return loc;
     else                              return -1;
  }
```

Character arrays (or string)

C++ supports two types of strings. The first is character arrays and the *null-terminated string* (also called C-string).

- Character array: An array whose elements are of type char.
- *null-terminated string* : sequence of zero or more characters enclosed in double quotation marks, the last character is always the null(null character is represented as '\0').

```
char studentName[26];
studentName = "Sara and Fatma"; //illegal
```

- assignment and comparison, are not allowed on string, then C++ provides a set of functions that can be used for string manipulation (found in *string* header file) while Most rules that apply to other arrays also apply to character arrays.

in the following summarizes these functions.

☒ Reading and Writing Strings

String Input

- the function **get** used to read strings that has two parameters:

1. string variable;
2. parameter specifies how many characters to read into the string variable.

Syntax:

cin.get(str, m);

Example:

```
char str[31];
cin.get(str, 31);
```

- The **getline** stream function use to read and store a line of input:

Example:

```
char textLine[100];
cin.getline(textLine, 100);
```

String Output

The output of strings by using an output stream variable, such as **cout** and **puts**.

Two Dimensional Arrays

Two-dimensional array is a collection of a fixed number of element arranged in rows and columns , where in all elements are of the same type. In two-dimensional arrays the data is provided in a table form.

Syntax

number of rows number of columns

Type arrayName[arraySize1][arraySize2];

type of elements array name

Example : Declares a two-dimensional array sales of 10 rows and 5 columns.

Ans:

double sales[10][5];

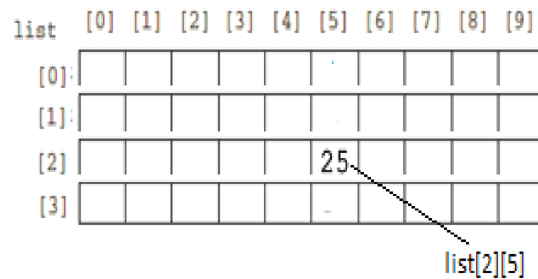
❑ Accessing Array elements

Syntax:

arrayName[indexExp1][indexExp2]

Where indexExp1 and indexExp2 are expressions nonnegative integer values. indexExp1 specifies the row position; indexExp2 specifies the column position.

The statement: list[5][2] = 25;



❑ Array Initialization

Like one-dimensional arrays, two-dimensional arrays can be initialized when they are declared.

Example: suppose the following:

int board[4][3] = {{2, 3, 1}, {15, 25, 13}, {20, 4, 7}, {11, 18, 14}};

row1 **row2** **row3** **row4**

note:

For number arrays, if all elements of a row are not specified, the unspecified elements are initialized to 0. In this case, at least one of the values must be given to initialize all the element of a row.

❑ Processing 2D arrays

A two-dimensional array can be processed in three ways:

1. Process the entire array such as initializing and printing the array.
2. Process a particular row of the array, called **row processing** such as finding the largest element in a row or finding the sum of a row.
3. Process a particular column of the array, called **column processing**, such as finding the largest element in a column or finding the sum of a column.

Each row and each column of a two-dimensional array is a one-dimensional array. Therefore, when processing a particular row or

column of a two-dimensional array, we use algorithms similar to those that process one-dimensional arrays.

To process row=5 :

Equivalent

```
for (col = 0; col<NUMBER_OF_COLUMNS; col++) ⇒ for (col = 0;col<NUMBER_OF_COLUMNS;col++)
   process matrix[row][col] ;                      process matrix[5][col];
```

- **To process column number 2 of matrix (col =2):**

Equivalent

```
for (row = 0; row<NUMBER_OF_ROWS; row++) ⇒ for (row = 0; row<NUMBER_OF_ROWS; row++)
   process matrix[row][col] ;                   process matrix[row][2];
```

- **Steps to initialize row 4 to 0:**

```
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
   matrix[row][col] = 0;
```

- **Steps to initialize the entire matrix to 0**

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
  for (col = 0; col < NUMBER_OF_COLUMNS; col++)
    matrix[row][col] = 0;
```

- **Steps to print the elements of matrix, one row per line:**

```
for (row = 0; row < NUMBER_OF_ROWS; row++) {
   for (col = 0; col < NUMBER_OF_COLUMNS; col++)
     cout <<  matrix[row][col] << " ";
   cout << endl;
}
```

- **Steps to inputs the data into row number 4 of matrix:**

```
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
  cin >> matrix[row][col];
```

- **Steps to input data into each element of matrix.**

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
  for (col = 0; col < NUMBER_OF_COLUMNS; col++)
    cin >> matrix[row][col];
```

- **Steps to loop finds the sum of row number 4 of matrix:**

```
sum = 0;
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
   sum = sum + matrix[row][col];
```

- **Steps to find the sum of each row separately.**

```
 for (row = 0; row < NUMBER_OF_ROWS; row++)  {
    sum = 0;
    for (col = 0; col < NUMBER_OF_COLUMNS; col++)
      sum = sum + matrix[row][col];
    cout << "Sum of row " << row + 1 << " = " << sum << endl;

  }
```

- **Steps to sum of each individual column:**

```
for (col = 0; col < NUMBER_OF_COLUMNS; col++){
   sum = 0;
   for (row = 0; row < NUMBER_OF_ROWS; row++)
     sum = sum + matrix[row][col];
   cout << "Sum of column " << col + 1 << " = " << sum << endl;
 }
```

- **Steps to determines the largest element in row 4:**

```
largest = matrix[row][0];   //Assume that the first element of
for (col = 1; col < NUMBER_OF_COLUMNS; col++)
  if (largest < matrix[row][col])
    largest = matrix[row][col];
```

- **Steps to determines the largest element in each row :**

```
for (row = 0; row < NUMBER_OF_ROWS; row++) {
     largest = matrix[row][0]; //Assume that the first element of the row is the largest.
    for (col = 1; col < NUMBER_OF_COLUMNS; col++)
      if (largest < matrix[row][col])
        largest = matrix[row][col];
    cout << "The largest element in row " << row + 1 << " = "<< largest
<< endl;
}
```

- **Steps to determines the largest element in each column:**

```
for (col = 0; col < NUMBER_OF_COLUMNS; col++){
    largest = matrix[0][col]; //Assume that the first element of the column is the largest.
    for (row = 1; row < NUMBER_OF_ROWS; row++)
      if (largest < matrix[row][col])
        largest = matrix[row][col];
      cout << "The largest element in column " << col + 1 << " = " <<
largest << endl;
```

Passing Two-Dimensional Arrays as Parameters to Functions

Two-dimensional arrays can be passed as parameters to a function, and they are passed by reference. C++ stores two-dimensional arrays in row order form, the compiler must know where one row ends and the next row begins. Thus, when declaring a two-dimensional array as a formal parameter, you can omit the size of the first dimension, but not the second; that is, you must specify the number of columns.

Suppose we have the following declaration:

```
const int NUMBER_OF_ROWS = 6;
const int NUMBER_OF_COLUMNS = 5;

// fuction to print the elements of matrix:
void printMatrix(int matrix[][NUMBER_OF_COLUMNS], int noOfRows){
   int row, col;
  for (row = 0; row < noOfRows; row++) {
      for (col = 0; col < NUMBER_OF_COLUMNS; col++)
        cout <<  matrix[row][col] << " ";
      cout << endl;
   }
}
```

// function to output sum of the elements of each row of a two dimensional array whose elements are of type int.

```
void sumRows(int matrix[][NUMBER_OF_COLUMNS], int noOfRows){
  int row, col;
  int sum;
  for (row = 0; row < noOfRows; row++){
     sum = 0;
      for (col = 0; col < NUMBER_OF_COLUMNS; col++)
       sum += matrix[row][col];
      cout << "Sum of row " << (row + 1) << " = " << sum << endl;
    }
 }
```

- **function to determines the largest element in each row:**

```
void largestInRows(int matrix[][NUMBER_OF_COLUMNS], int
noOfRows{
     int row, col;
     int largest;
     for (row = 0; row < noOfRows; row++) {
           largest = matrix[row][0];
            for (col = 1; col < NUMBER_OF_COLUMNS; col++)
              if (largest < matrix[row][col])
                largest = matrix[row][col];
              cout << "The largest element of row "<< (row+1)<< "=" <<
largest <<endl;
        }
      }
```

Multi Dimensional Arrays

General form for multi dimensional array:

Type arrayName [arraySize1] [arraySize2] … [arraySizen];

Where

- **Type** is any known type
- **arrayName** is array name
- arraySizei is the number of elements for the dimension i (i=1..n)

Suppose we need to store phone directory for N person, we need program to:

1. Let us to enter name and phone number
2. Display phone number for any person