Chapter 8: Arrays

An **array** is a collection of elements of the same type placed in contiguous memory locations. Arrays are used to store a set of values of the same type under a single variable name. Each element in an array can be accessed using its position in the list called index number or subscript.

The syntax for declaring an array in C++ is as follows.


```
data_type array_name[size];
```

Eg: `int num[10];`

This statement declares an array named `num` that can store 10 integer numbers.

For a single dimensional array, the total size allocated can be computed using the following formula:

total_bytes = sizeof(array_type) × size_of_array

The array in the above example require 4 x 10 = 40 Bytes

Array elements can be initialised in their declaration statements in the same manner as in the case of variables, except that the values must be included in braces, as shown in the following example:

```
int score[5] = {98, 87, 92, 79, 85};
```

Algorithm for selection sort

Step 1. Start
Step 2. Accept a value in N as the number of elements of the array
Step 3. Accept N elements into the array AR
Step 4. Repeat Steps 5 to 9, (N – 1) times
Step 5. Assume the first element in the list as the smallest and store it in MIN and its position in POS
Step 6. Repeat Step 7 until the last element of the list
Step 7. Compare the next element in the list with the value of MIN. If it is found smaller, store it in MIN and its position in POS
Step 8. If the first element in the list and the value in MIN are not the same, then swap the first element with the element at position POS
Step 9. Revise the list by excluding the first element in the current list
Step 10. Print the sorted array AR
Step 11. Stop

Algorithm for bubble sort

Step 1. Start
Step 2. Accept a value in N as the number of elements of the array
Step 3. Accept N elements into the array AR
Step 4. Repeat Steps 5 to 7, (N - 1) times
Step 5. Repeat Step 6 until the second last element of the list
Step 6. Starting from the first position, compare two adjacent elements in the list. If they are not in proper order, swap the elements.
Step 7. Revise the list by excluding the last element in the current list.
Step 8. Print the sorted array AR
Step 9. Stop

Algorithm for Linear Search

Step 1. Start
Step 2. Accept a value in N as the number of elements of the array
Step 3. Accept N elements into the array AR
Step 4. Accept the value to be searched in the variable ITEM
Step 5. Set LOC = -1
Step 6. Starting from the first position, repeat Step 7 until the last element
Step 7. Check whether the value in ITEM is found in the current position. If found then store the position in LOC and Go to Step 8, else move to the next position.
Step 8. If the value of LOC is less than 0 then display "Not Found", else display the value of LOC + 1 as the position of the search value.
Step 9. Stop

Algorithm for binary search

Step 1. Start
Step 2. Accept a value in MAX as the number of elements of the array
Step 3. Accept MAX elements into the array LIST
Step 4. Accept the value to be searched in the variable ITEM
Step 5. Store the position of the first element of the list in FIRST and that of the last in LAST
Step 6. Repeat Steps 7 to 11 While (FIRST <= LAST)
Step 7. Find the middle position using the formula (FIRST + LAST)/2 and store it in MIDDLE
Step 8. Compare the search value ITEM with the element at the MIDDLE of the list
Step 9. If the MIDDLE element contains the search value ITEM then stop search, display the position and go to Step 12.
Step 10. If the search value is smaller than the MIDDLE element
Then set LAST = MIDDLE - 1
Step 11. If the search value is larger than the MIDDLE element
Then set FIRST = MIDDLE + 1
Step 12. Stop

Linear search method	Binary search method
• The elements need not be in any order	• The elements should be in sorted order
• Takes more time for the process	• Takes very less time for the process
• May need to visit all the elements	• All the elements are never visited
• Suitable when the array is small	• Suitable when the array is large

Two dimensional (2D) arrays

A two dimensional array is an array in which each element itself is an array. For instance, an array AR[m][n] is a 2D array containing **m** rows and **n** columns.

The general form of a two dimensional array declaration in C+ + is as follows :

```
data_type array_name[rows][columns];
```

where `data_type` is any valid data type of C++ and elements of this 2D array will be of this type. The `rows` refers to the number of rows in the array and `columns` refers to the number of columns in the array. The indices (subscripts) of rows and columns, start at 0 and ends at (rows–1) and (columns–1) respectively. The following declaration declares an array named marks of size 5 × 4 (5 rows and 4 columns) of type integer.

int marks[5][4];