Lecture 1: Introduction to Data Structures using C

In this lecture, the following topics are covered:

- ❑ The concept of data structure
- ❑ Classification of Data Structures
- ❑ Operations on Data Structures
- ❑ Arrays (one dimensional)

1. Introduction

- A computer program should undoubtedly give correct results, but along with that it should also run efficiently.
- A program is said to be efficient when it executes in minimum time and with minimum memory space. In order to write efficient programs, we need to apply certain data management concepts.
- The concept of data management is a complex task that includes activities like data collection, organization of data into appropriate structures, and developing and maintaining routines for quality assurance.
- A data structure is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.
- Data structures are used in almost every program or software system. Some common examples of data structures are arrays, linked lists, queues, stacks, binary trees, and hash tables.
- Data structures are widely applied in the following areas:

Compiler design Operating system Statistical analysis package DBMS Numerical analysis Simulation Artificial intelligence Graphics

2. Classification of Data Structures

Data structures can be classified into two categories: linear and non-linear data structures.

Linear Data Structures

- Elements of a data structure are stored in a linear or sequential order.
- Examples include arrays, linked lists, stacks, and queues.
- Linear data structures can be represented in memory in two different ways:
    - One way is to have a linear relationship between elements by means of sequential memory locations (Arrays).
    - The other way is to have a linear relationship between elements by means of links (Linked lists).

Non-Linear Data Structures

- if the elements of a data structure are not stored in a sequential order, then it is a non-linear data structure.
- Examples include trees and graphs.

3. Operations On Data Structures

This section discusses the different operations that can be performed on the various data structures previously mentioned.

Traversing: It means to access each data item exactly once so that it can be processed. For example, to print the names of all the students in a class.

Searching: It is used to find the value or location of one or more data items that satisfy the given constraint. Such a data item may or may not be present in the given collection of data items. For example, to find the names of all the students who passed the mathematics exam.

Inserting: It is used to add new data items to the given list of data items. For example, to add the details of a new student who has recently joined the course.

Deleting: It means to remove (delete) a particular data item from the given collection of data items. For example, to delete the name of a student who has left the course.

Sorting: Data items can be arranged in some order like ascending order or descending order depending on the type of application. For example, arranging the names of students in a class in an alphabetical order, or calculating the top three winners by arranging the participants' scores in descending order and then extracting the top three.

Linear Data Structures (1)

Arrays

*An array is **a collection of similar data elements**. These data elements have the same data type. The elements of the array are stored in consecutive memory locations and are referenced by an* **index**.

1. Declaration of Arrays

- In C, every variable must be declared before it is used. The same concept holds true for array variables.
- An array must be declared before being used.
- Declaring an array means specifying the following:
  - ✓ **Data type**—the kind of values it can store, for example, **int**, **char**, **float**, **double**.
  - ✓ **Name**—to identify the array.
  - ✓ **Size**—the maximum number of values that the array can hold.

- Arrays are declared using the following syntax:

type array_name[size];

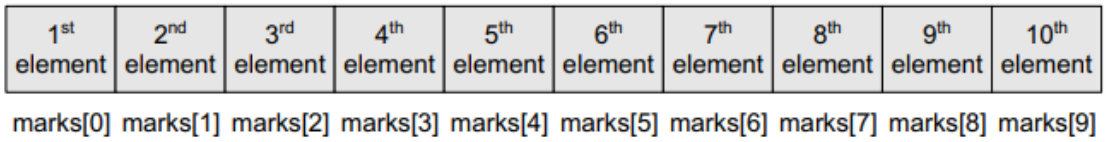
  - The **type** can be either **int**, **float**, **double**, **char**, or **any other valid data type**.
  - The number within brackets indicates the **size** of the array, i.e., the maximum number of elements that can be stored in the array.

- The statement **int marks [10];** declares **marks** to be an array containing **10** elements of **int** type.

In C, the array index starts from zero.

- The first element is stored at index **0,** the second element at index **1**, the third element at index **2** and so on.
- The last element of the array is located at index **n-1**, where **n** is the size of the array.
- For the **marks** array declared above, the first element is **marks [0],** the second element is **marks [1]** while the last element is **marks [9].**

- Note that the numbers 0, 1, 2, 3, ….9 written within square brackets are the indexes.
- In the memory, the **marks** array will be stored as shown in the following figure:



- The size of array is fixed. Once declared, the size of the array can't be changed to accommodate more elements.
- In arrays, accessing elements is done randomly.

2. Accessing the elements of an array

- Storing related data items in a single array enables the programmers to develop concise and efficient programs. But there is no single function that can operate on all the elements of an array.
- **To access all the elements, we must use a loop**. That is, we can access all the elements of an array by changing the value of the index of the array.
- The index must be **an integer value** or **an expression that evaluates to an integer value**.
- Now to process all the elements of the array, we use a loop as shown in following snippet of code:

```
// Set each element of the array to -1
int i, marks[10];
for(i=0;i<10;i++)
        marks[i] = -1;
```

- The above code accesses every individual element of the **marks** array and sets its value to **–1**.
- In the **for** loop, first the value of **marks [0]** is set to **–1**, then the value of the index (i) is incremented and the next value, that is, **marks [1]** is set to **–1**. The procedure continues until all the **10** elements of the array are set to **–1.**

3. Storing Values in Arrays

When we declare an array, we are just allocating space for its elements; no values are stored in the array. There are **three ways to store values in an array**:

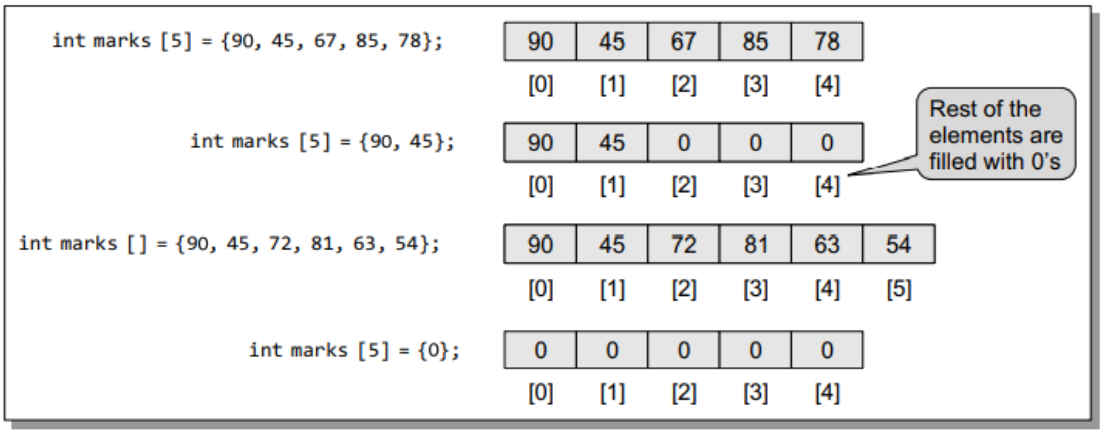
- **to initialize the array elements during declaration**.
- **to input values for individual elements from the keyboard**.
- **to assign values to individual elements**.

a) Initializing Arrays during Declaration

- The elements of an array can be initialized at the time of declaration, just as any other variable.
- When an array is initialized, we need to provide a value for every element in the array. Arrays are initialized by writing,

  type array_name[size] = {list of values};
- Note that the values are written within curly brackets and every value is separated by a comma.
- **It is a compiler error to specify more values than there are elements in the array.**
- When we write, **int marks [5] = {90, 82, 78, 95, 88};** An array with the name **marks** is declared that has enough space to store five elements. The first element, that is, **marks [0]** is assigned the value **90**. Similarly, the second element of the array, that is **marks [1]**, is assigned **82**, and so on.
- **While initializing the array at the time of declaration, the programmer may omit the size of the array**. For example, **int marks [] = {98, 97, 90};** The above statement is absolutely legal. Here, the compiler will allocate enough space for all the initialized elements.

- Note that if the number of values provided is less than the number of elements in the array, the un-assigned elements are filled with zeros. This is shown in following diagram.



b) Inputting Values from the Keyboard

An array can be initialized by inputting values from the keyboard. In this method, a while/do–while or a for loop is executed to input the value for each element of the array. For example, look at the following code:

```
int i, marks[10];
for(i=0;i<10;i++)
       scanf("%d", &marks[i]);
```

In the code, we start at the index i at 0 and input the value for the first element of the array. Since the array has 10 elements, we must input values for elements whose index varies from 0 to 9.

c) Assigning Values to Individual Elements

The third way is to assign values to individual elements of the array by using the assignment operator. Any value that evaluates to the data type as that of the array can be assigned to the individual array element. A simple assignment statement can be written as **marks** [3] = 100.

We can also use a loop to assign **a pattern of values** to the array elements. For example, if we want to fill an array with even integers (starting from 0), then we will write the code as shown in this snippet of code:

```
// Fill an array with even numbers
int i,arr[10];
for(i=0;i<10;i++)
       arr[i] = i*2;
```

4. Operations on Arrays

1. Traversing an Array

- Traversing an array means accessing each and every element of the array for a specific purpose.
- Traversing the data elements of an **array A** can include:
    - printing every element,
    - counting the total number of elements, or performing any process on these elements.
- Since, array is a linear data structure (because all its elements form a sequence), traversing its elements is very simple and straightforward.

The pseudocode for an array traversing is shown in Fig.

```
Step 1: [INITIALIZATION] SET I = lower_bound
Step 2: Repeat Steps 3 to 4 while I <= upper_bound
Step 3:      Apply Process to A[I]
Step 4:      SET  I = I + 1
        [END OF LOOP]
Step 5: EXIT
```

Figure 1. The pseudocode for traversing the array A.

Exercises

1. Write a C program to read and display n numbers using an array?
2. Write a C program to find the mean of n integer numbers using arrays?
3. Write a C program to print the position of the smallest number of n numbers using arrays?
4. Write a C program to find the second largest of n integer numbers using an array?
5. Write a C program to enter n number of digits. Form a number using these digits?
6. Write a C program to find whether the array of integers contains a duplicate number?

2. Inserting an Element in an Array

If an element has to be inserted at the end of an existing array, then the task of insertion is quite simple. We just have to add 1 to the upper_ bound and assign the value. Here, we assume that the memory space allocated for the array is still available.

For example, if an array is declared to contain 10 elements, but currently it has only 8 elements, then obviously there is space to accommodate two more elements. But if it already has 10 elements, then we will not be able to add another element to it. Figure 3.13 shows an algorithm to insert a new element to the end of an array