BASIC TERMINOLOGY

Elementary Data Structure Organization

- While a data item that does not have subordinate data items is categorized as an elementary item, the one that is composed of one or more subordinate data items is called a group item.
- For example, a student's name may be divided into three sub-items—first name, middle name, and last name—but his roll number would normally be treated as a single item.
- A record is a collection of data items. For example, the name, address, course, and marks obtained are individual data items.
- But all these data items can be grouped together to form a record.

BASIC TERMINOLOGY

Elementary Data Structure Organization

- A file is a collection of related records.
- For example, if there are 60 students in a class, then there are 60 records of the students.
- All these related records are stored in a file.
- Similarly, we can have a file of all the employees working in an organization, a file of all the customers of a company, a file of all the suppliers, so on and so forth.
- Moreover, each record in a file may consist of multiple data items but the value of a certain data item uniquely identifies the record in the file. Such a data item K is called a primary key, and the values K1, K2 ... in such field are called keys or key values.

BASIC TERMINOLOGY

Elementary Data Structure Organization

- For example, in a student's record that contains roll number, name, address, course, and marks obtained, the field roll number is a primary key.
- Rest of the fields (name, address, course, and marks) cannot serve as primary keys, since two or more students may have the same name, or may have the same address (as they might be staying at the same place), or may be enrolled in the same course, or have obtained same marks.

CLASSIFICATION OF DATA STRUCTURES

- Data structures are generally categorized into two classes: primitive and non-primitive data structures.
- Primitive data structures are the fundamental data types which are supported by a programming language.
- Some basic data types are integer, real, character, and boolean.
- The terms 'data type', 'basic data type', and 'primitive data type' are often used interchangeably.
- Non-primitive data structures are those data structures which are created using primitive data structures.
- Examples of such data structures include linked lists, stacks, trees, and graphs.

CLASSIFICATION OF DATA STRUCTURES

Linear and Non-linear Structures

- Non-primitive data structures can further be classified into two categories: linear and non-linear data structures.
- If the elements of a data structure are stored in a linear or sequential order, then it is a linear data structure.
- 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 to a linear relationship between elements by means of sequential memory locations.
- The other way is to have a linear relationship between elements by means of links.

CLASSIFICATION OF DATA STRUCTURES

Linear and Non-linear Structures

- However, if the elements of a data structure are not stored in a sequential order, then it is a non-linear data structure.
- The relationship of adjacency is not maintained between elements of a non-linear data structure.
- Examples include trees and graphs. C supports a variety of data structures.
- We will now introduce all these data structures and they would be discussed in detail in subsequent chapters.

CLASSIFICATION OF DATA STRUCTURES

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 (also known as the subscript).
- In C, arrays are declared using the following syntax: type name[size];
- For example, int marks[10]; The above statement declares an array marks that contains 10 elements.
- In C, the array index starts from zero.
- This means that the array marks will contain 10 elements in all.
- The first element will be stored in marks[0], second element in marks[1], so on and so forth.
- Therefore, the last element, that is the 10th element, will be stored in marks[9]. In the memory, the array will be stored as shown in Fig. 2.1.

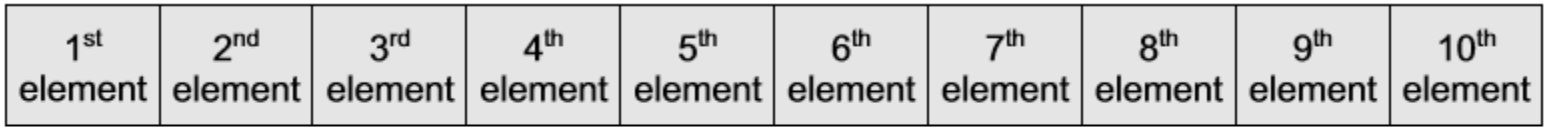

Figure 2.1 Memory representation of an array of 10 elements

CLASSIFICATION OF DATA STRUCTURES

Arrays

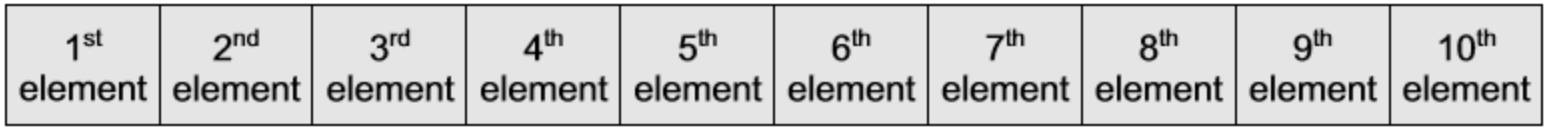

Figure 2.1 Memory representation of an array of 10 elements

- Arrays are generally used when we want to store large amount of similar type of data.
- But they have the following limitations:
  - Arrays are of fixed size.
  - Data elements are stored in contiguous memory locations which may not be always available.
  - Insertion and deletion of elements can be problematic because of shifting of elements from their positions.
- However, these limitations can be solved by using linked lists.
- We will discuss more about arrays in Chapter 3.

Linked Lists

- A linked list is a very flexible, dynamic data structure in which elements (called nodes) form a sequential list.
- In contrast to static arrays, a programmer need not worry about how many elements will be stored in the linked list.
- This feature enables the programmers to write robust programs which require less maintenance.
- In a linked list, each node is allocated space as it is added to the list.

Figure 2.2 Simple linked list

Linked Lists

- Every node in the list points to the next node in the list.
  - Therefore, in a linked list, every node contains the following two types of data: The value of the node or any other data that corresponds to that node
  - A pointer or link to the next node in the list
- The last node in the list contains a NULL pointer to indicate that it is the end or tail of the list.
- Since the memory for a node is dynamically allocated when it is added to the list, the total number of nodes that may be added to a list is limited only by the amount of memory available.
- Figure 2.2 shows a linked list of seven nodes.

Figure 2.2 Simple linked list