DATA STRUCTURES

INTRODUCTION

Data Structures

- *A Data Structure is an arrangement of data either in computer's memory or on the disk storage*
    - *enables efficient access and modification.*
- ***A Data Structure is a way of organizing, storing and retrieving data and their relationship with each other in memory of computer***

Data Structures

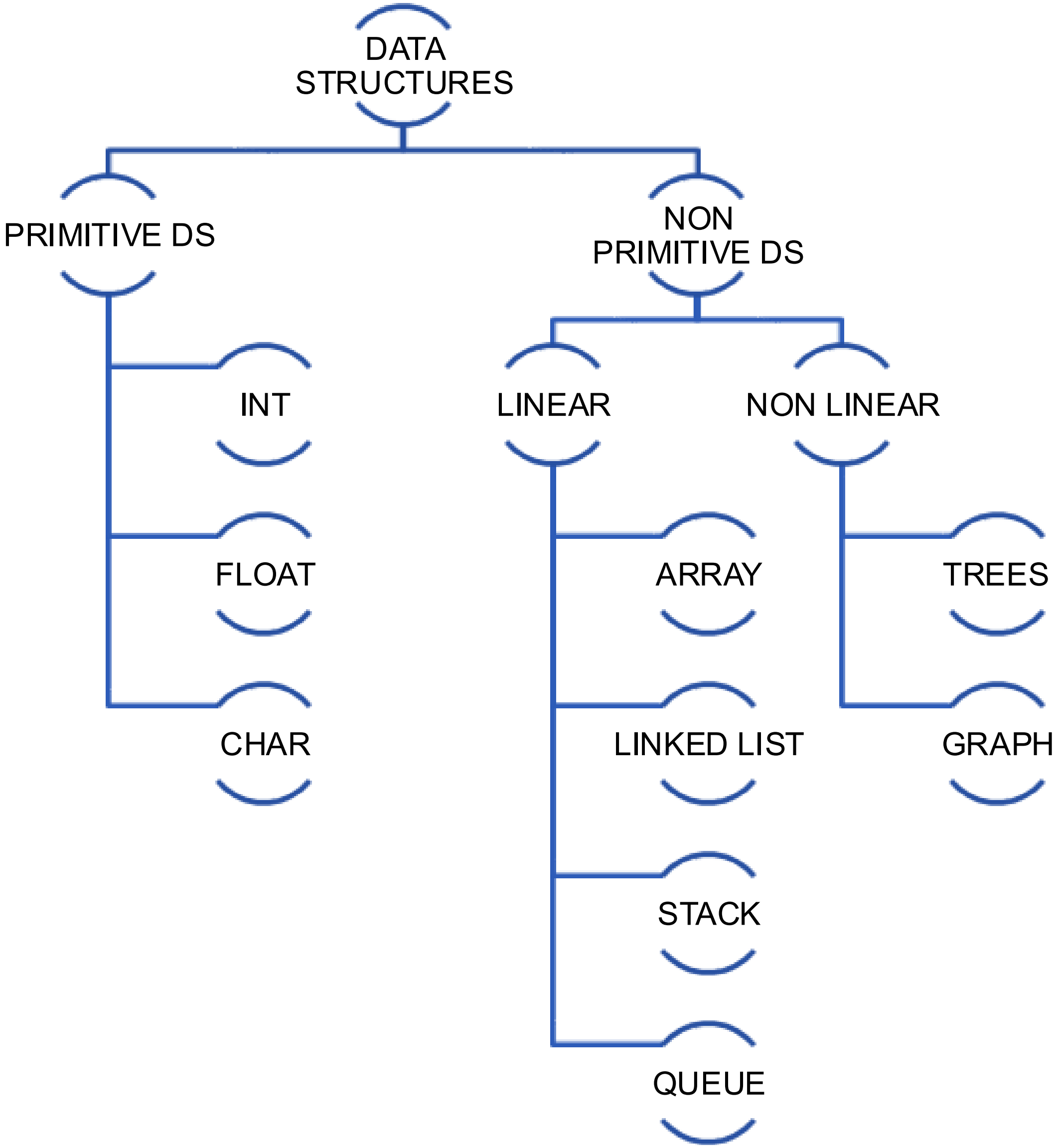
- ***Let we take example of Residence***

APPLICATIONS OF DATA STRUCTURES

Data structures are widely applied in areas like:

- *Compiler design*
- *Operating system*
- *Statistical analysis package*
- *DBMS*
- *Numerical analysis*
- *Simulation*
- *Artificial Intelligence*

CLASSIFICATIONS OF DATA STRUCTURES



CLASSIFICATIONS OF 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:**

  arrays, linked lists, stacks, and queues.

- If the elements of a data structure are not stored in a sequential order, then it is a **non-linear data structure**. **Examples:**

  trees and graphs

UNIT - I

LISTS

Abstract Data Types (ADTs) – List ADT – Array-based implementation – Linked list implementation – Singly linked lists – Circularly linked lists – Doubly-linked lists – Applications of lists – Polynomial ADT – Radix Sort – Multilists

ADT

ADT stands for Abstract Data Type

Abstract Data Type is a mathematical model and set of operations defined on that model, providing only the essentials and hiding the details

Types:

List ADT, Stack ADT, Queue ADT

List ADT

- **A list is a sequence of elements of a given type**

- **A list can be implemented in two ways**
  - **Array Based implementation**
  - **Pointer Based implementation (or) Linked List Implementation**
    - **Singly Linked List**
    - **Doubly Linked List**
    - **Circular Linked List**

List – Array Based implementation

- **An array is a collection of similar data elements.**

- **Elements of arrays are stored in consecutive memory locations and are referenced by an index (or subscript).**

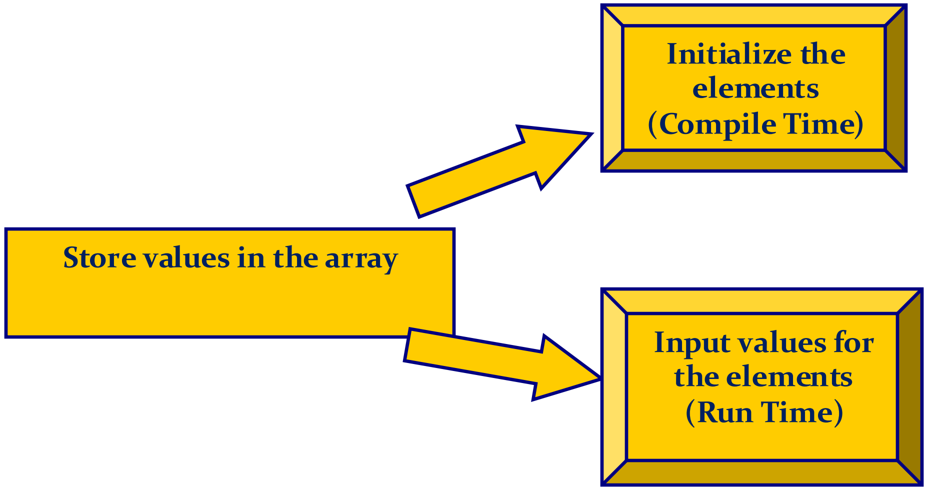
- **Arrays are declared using the following syntax:**

type name[size];

1st element	2nd element	3rd element	4th element	5th element	6th element	7th element	8th element	9th element	10th element
marks[0]	**marks[1]**	**marks[2]**	**marks[3]**	**marks[4]**	**marks[5]**	**marks[6]**	**marks[7]**	**marks[8]**	**marks[9]**

List – Array Based implementation

Storing Values in Arrays


Store values in the array
Initialize the elements (Compile Time)
Input values for the elements (Run Time)

Initializing Arrays during declaration

int marks [5] = {90, 98, 78, 56, 23};

Inputting Values from Keyboard

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

List – Array Based implementation

Accessing Elements of an Array

- To access all the elements of an array, we must use a loop.
- That is, we can access all the elements of an array by varying the value of the subscript into the array.

```
int i, marks[10];
for(i=0;i<10;i++)
   printf(“%d”, marks[i]);
```

List – Array Based implementation

Inserting an Element in an Array

Algorithm to insert a new element to the end of an array

```
Step 1: Set upper_bound = upper_bound + 1
Step 2: Set A[upper_bound] = VAL
Step 3; EXIT
```

Algorithm INSERT(A, N, POS, VAL) to insert an element VAL at position POS

```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:         SET A[I + 1] = A[I]
Step 4:         SET I = I – 1
      [End of Loop]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

List – Array Based implementation

Deleting an Element from an Array

Algorithm to delete an element from the end of the array

```
Step 1: Set upper_bound = upper_bound - 1
Step 2: EXIT
```

Algorithm DELETE(A, N, POS) to delete an element at POS

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N-1
Step 3:         SET A[I] = A[I + 1]
Step 4:         SET I = I + 1
        [End of Loop]
Step 5: SET N = N - 1
Step 6: EXIT
```