Data Structures

Prof. Anand Jain, anand.jain@pccoepune.org

Prerequisite

- Computer Fundamentals
- C Programming

Course Outcomes

After learning the course, the students should be able to:

1. To demonstrate data structures linked list, stack and queue.

2. To describe searching and sorting techniques.

3. To implement tree, graph data structures.

4. To apply design principles and concepts for Data Structure to solve problems.

Unit 01 : Introduction to Data Structure

Fundamentals of Data Structure

- **Data Structure Definition**
  - A data structure is a system used to store data, keep it organized, and enable easy modification and access. A data structure refers to a group of data values, how they relate to each other, and the operations or functions that can be carried out on them.

- **Need of Data Structure**
  - To organize the data in main memory (RAM)
  - To improve the efficiency of computer while performing operations
  - To handle large amount of data with desired processing speed
  - Data Search

Classification of Data Structure

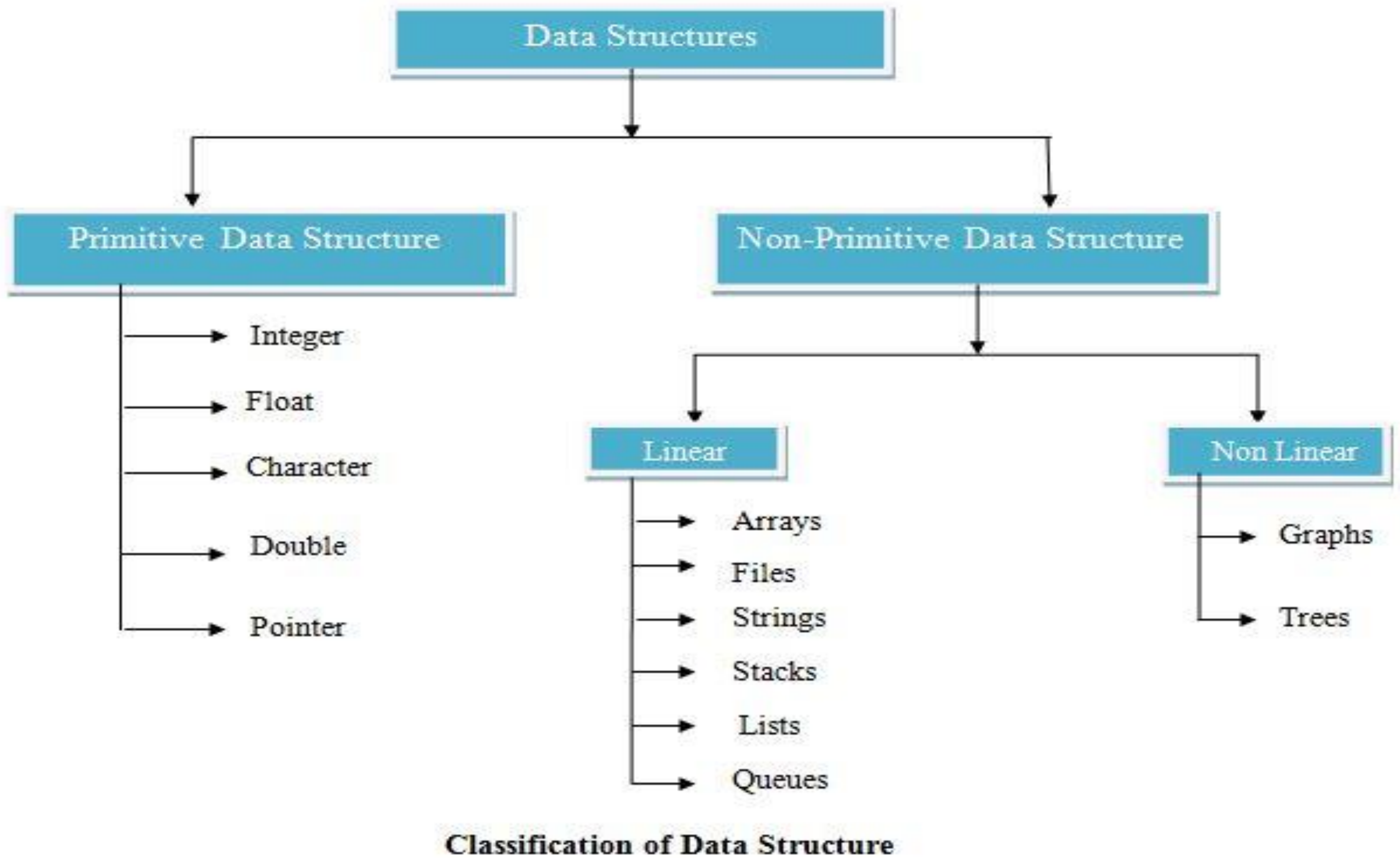


Image Source : https://techieshouts.com/data-structures-introduction/

Classification of Data Structure

- **Primitive Data Structure**
  - The data structures that are directly operated upon by machine level instructions are called as primitive data structures. These are predefined types of data, which are supported by the programming language. For example, integer, character, and string are all primitive data types. Programmers can use these data types when creating variables in their programs. For example, a programmer may create a variable called "lastname" and define it as a string data type. The variable will then store data as a string of characters.

Classification of Data Structure

- **Non-Primitive Data Structure**
  - Derived from primitive data structures
  - Not defined by the programming language but are created by the programmer
  - emphasize on structuring of a group of homogeneous or heterogeneous data items
  - further divided into Linear and Non-Linear data structure
  - Examples : Array, Stack, Tree, File etc

Classification of Data Structure

- **Linear Data Structure :** Data structure whose element(objects) are sequential and ordered in a way so that:
  - there is only one first element and has only one next element
  - there is only one last element and has only one previous element
  - all other elements have a next and a previous element

- A Linear data structure have data elements arranged in sequential manner and each member element is connected to its previous and next element. Such data structures are easy to implement as computer memory is also sequential.

- Examples of linear data structures are List, Queue, Stack, Array etc.

Classification of Data Structure

- **Non-Linear Data Structure :** Nonlinear data structures are those data structures in which data items are not arranged in a sequence.

- A non-linear data structure has no set sequence of connecting all its elements and each element can have multiple paths to connect to other elements.

- Such data structures supports multi-level storage and often cannot be traversed in single run. Such data structures are not easy to implement but are more efficient in utilizing computer memory.

- Examples of non-linear data structures are Tree, BST, Graphs etc.

Classification of Data Structure

❖ **Linear Vs. Non-Linear Data Structure**

Sr. No.	Key	Linear Data Structures	Non-linear Data Structures
1	Data Element Arrangement	In linear data structure, data elements are sequentially connected and each element is traversable through a single run.	In non-linear data structure, data elements are hierarchically connected and are present at various levels.
2	Levels	In linear data structure, all data elements are present at a single level.	In non-linear data structure, data elements are present at multiple levels.
3	Implementation complexity	Linear data structures are easier to implement.	Non-linear data structures are difficult to understand and implement as compared to linear data structures.
4	Traversal	Linear data structures can be traversed completely in a single run.	Non-linear data structures are not easy to traverse and needs multiple runs to be traversed completely.
5	Memory utilization	Linear data structures are not very memory friendly and are not utilizing memory efficiently.	Non-linear data structures uses memory very efficiently.
6	Time Complexity	Time complexity of linear data structure often increases with increase in size.	Time complexity of non-linear data structure often remain with increase in size.
7	Examples	Array, List, Queue, Stack.	Graph, Map, Tree.

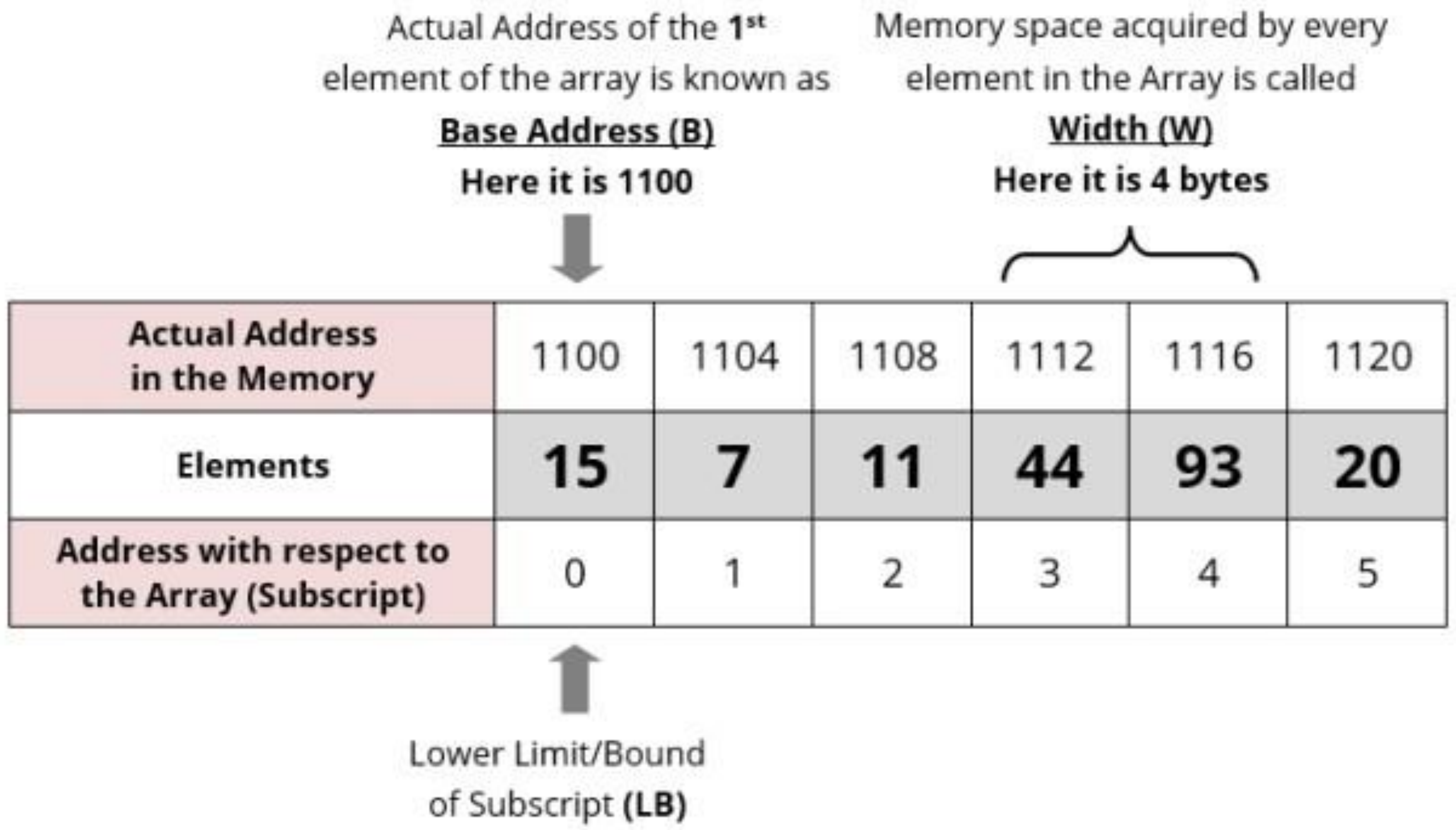
Array

- An array is finite, ordered and collection of homogeneous data elements.
- Used to store group of data together in one place
- All the elements of an array are stored in linear order
- Syntax in C : datatype arrayName[arraySize];
- For ex. : int num[10];

- **Terminologies**
  - Size
  - Type
  - Base
  - Index
  - Range of Indices

Array

❖ **One Dimensional Array Representation**



Actual Address of the 1st element of the array is known as **Base Address (B)**
Here it is 1100

Memory space acquired by every element in the Array is called **Width (W)**
Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

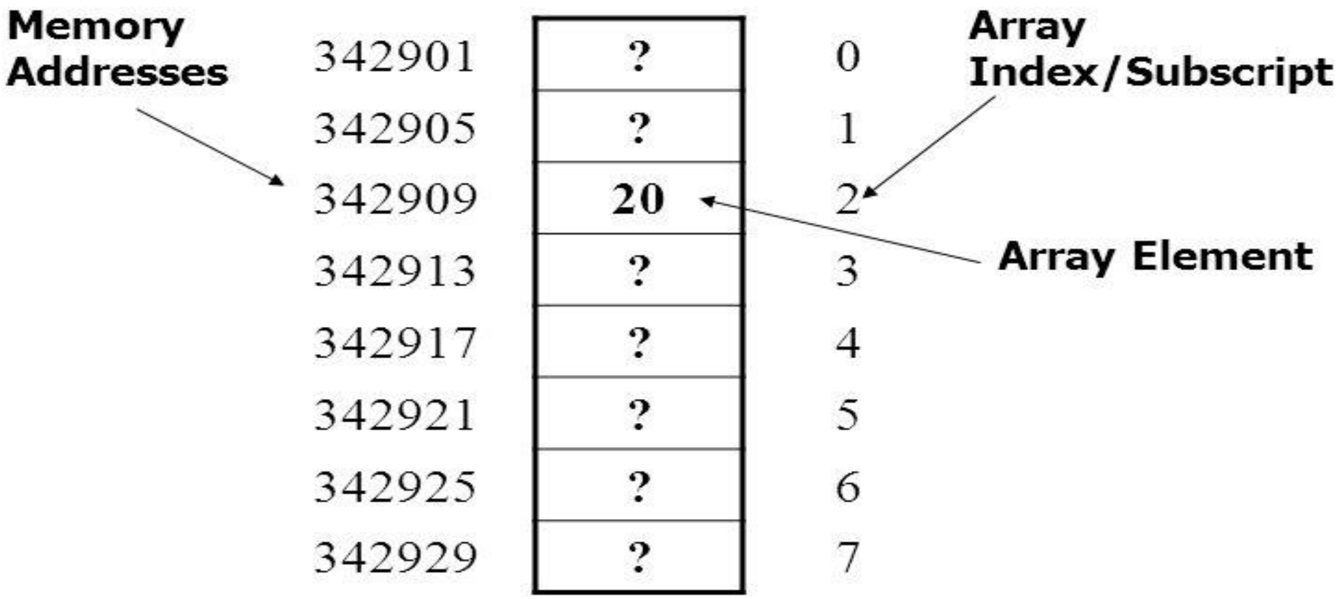
Lower Limit/Bound of Subscript **(LB)**

Array

Visual representation of a One-dimensional Array

```
int x[8];

x[2] = 20;
```



Note: Index starts with 0, not with 1

Array

- **Operations on Array**
  - Traverse : Display all the elements of array one by one
  - Insert : Adds an element at the given index
  - Delete : Deletes an element at the given index
  - Search : Searches an element using the given index or by the value
  - Update : Updates an element at the given index
  - Sort : Sort array elements in ascending or descending order
  - Merge : Combining two (or more) arrays into one array

Array

- **Insert Operation on Array :** This operation is used to insert an element into an array, provided array is not full.
- **Algorithm**
  - 1. Get the element value which needs to be inserted.
  - 2. Get the position value.
  - 3. Check whether the position value is valid or not.
  - 4. If it is valid,
    - Shift all the elements from the last index to position index by 1 position to the right.
    - insert the new element in arr[position]
  - 5. Otherwise,
    - Invalid Position

Array

- **Delete Operation on Array :** This operation is used to delete a particular element from an array, provided array is not empty. The element will be deleted by pushing the tail (part of array after the element which is to be deleted) one stroke up.

- **Algorithm**
  - 1. Find the given element in the given array and note the index.
  - 2. If the element found,
    - Shift all the elements from index + 1 by 1 position to the left
  - 3. Otherwise, print "Element Not Found"

Array

- **Search Operation on Array :** This operation is applied to search an element in the array.

- **Binary Search Algorithm**
  - At every step, consider the array between low and high indices
  - Calculate the mid index.
  - If the element at the mid index is the key, return mid.
  - If the element at mid is greater than the key, then change the index high to mid - 1.
  - The index at low remains the same.
  - If the element at mid is less than the key, then change low to mid + 1. The index at high remains the same.
  - When low is greater than high, the key doesn’t exist and -1 is returned.

Array

- **Insert Operation on Array :** This operation is used to insert an element into an array, provided array is not full.

- **Algorithm**
  - 1. Get the element value which needs to be inserted.
  - 2. Get the position value.
  - 3. Check whether the position value is valid or not.
  - 4. If it is valid,
    - Shift all the elements from the last index to position index by 1 position to the right.
    - insert the new element in arr[position]
  - 5. Otherwise,
    - Invalid Position

Thank You !