Name of Teacher: Miss Radhika M. Patil

Class: B.Sc. Computer Science (Entire)- II **Semester : 4**

Course Title: Introduction to Data Structure using C++

UNIT 1: Introduction to Data structure and Linear Data Structures

- **Introduction to Data Structure:**

- The data structure name indicates itself that organizing the data in memory.
- A data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
- Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently.
- A data structure is a specialized format for organizing, processing, retrieving and storing data.
- Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artificial intelligence, Graphics and many more.
- Data structures are the building blocks of any program or the software.
- Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned.

- **Some Definitions:**

1. Data Type
2. Data Object
3. Data Structure

1. Data Type:

- Data type is the term used to describe the information type and can be processed by the computer and supported by the programming language.
- It is the collection of values and operations possible on those values.
- The possible operations for data type are addition, subtraction, multiplication etc.
- It can be also defined as a term which refers to the kind of data that the variables may hold .
- Data type is a classification identifying one of various types of data, such as floating-point, integer, or Boolean, that determines the possible values for that type

- **Abstract Data Type (ADT)**

- The concept of data type is extended to the real world object and user defined data types such as array, structure, class, union,etc. because the large program deals with collection of data types.
- There will be a few common operations on any collection.

ADT Definition:

- ADT can be defined as a collection of data values and functions which operated on these data values.
- It is implementation independent and isolates the program for representation.

- **Advantages of ADT:**

1. It is language independent
2. It prevents the direct manipulation of data values through functions defined.
3. The different modules can be used by different persons for different purposes with small changes.
4. ADTs encourage the modularity of programming.
5. By using ADT we can change implementation quickly.
6. ADT is only concerned with the data or properties and operations.

Example:

Students can be characterized with many properties such as roll no., name, marks in differents subjects, date of birth, address, class, previous qualification and so on.

There may be different possible operations which can be performed on these properties of student.

1. Insert record of students
2. Display records of student
3. Edit the information
4. Print result sheet etc.

2. Data Object:

- Data Object refers to a set of elements ‘D’ which may be finite or infinite.
- If the set is infinite, we have ti decide some mechanism to represent such set in memory since available memory is limited.

Example:

1) A set integer number is infinite

i.e. D= {……-4, -3, -2, -1, 0,+1, +2, +3, +4……}

2) A set of alphabets is finite

i.e. D= { ‘a’, ‘b’, ‘c’,…….’z’}

3. Data Structure

- Data Structure is a way of collecting and organizing of data in such a way that we can perform the operations on these data in an effective way.
- Data Structure is organization of data elements in computer memory.
- The mathematical model to organize the data in computer memory and the methods to process them are collectively called as Data Structure.
- Data Structures are basic building blocks of program.

Definition:

Data Structure is consisting of a set of domain 'D', a set of Axioms and a set of function 'F'

i.e. **the triplet (D,F,A)= Data Structure.**

Where

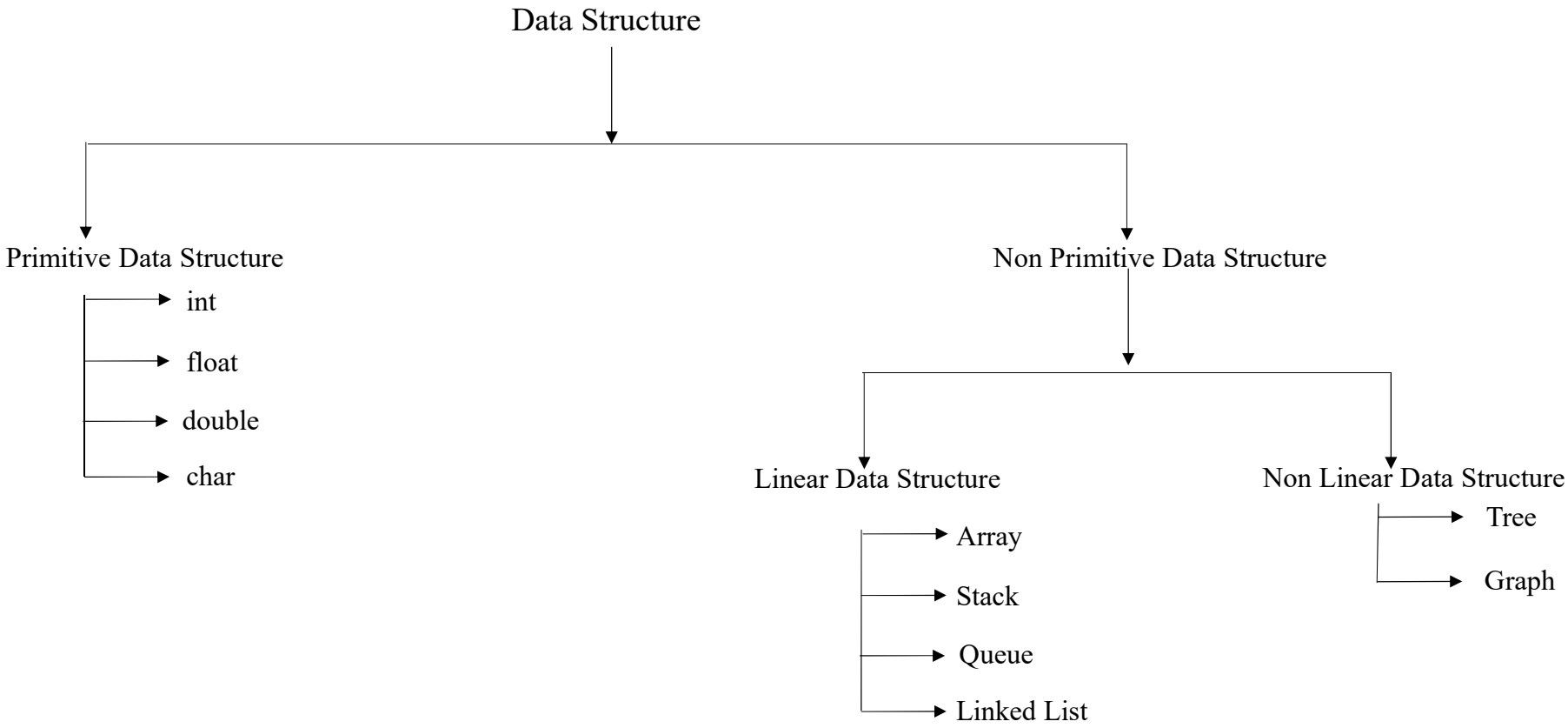
D denotes Data Object

F denotes operations that can be carried out on data object.

A denotes the rule or principle of operation.

- **Classification of Data Structure:**

  Data Structure can be classified as



1. Primitive Data Structure:

- Primitive data structures are used to represent the standard data types.
- Primitive data structures are used to represent a single value.
- e.g. int, float, double, char, Boolean,etc.

2. Non Primitive Data Structure:

- Non Primitive Data Structure can be constructed with the help of any one of the primitive data structures.
- They have a specific functionality.

It can be further classified as

1. Linear Data Structure
2. Non Linear Data Structure

1. Linear Data Structure:

- It is a data structure in which data elements are organized in some **sequence** is called linear data structure.
- e.g. Array, Stack, Queue and Linked List
- Various operations are possible only in sequence i.e. you can't insert the element directly into some location without traversing (visiting) the previous element of data structure.

2. Non Linear Data Structure:

- It is a data structure in which data elements are organized in **arbitrary order** is called non linear data structure.
- They are not organized in any sequence.
- e.g. Tree, Graph

- **Common Operations on Data Structure:**

The basic operations that are performed on data structures are as follows:

1. **Insertion:** Insertion means addition of a new data element in a data structure.
2. **Deletion:** Deletion means removal of a data element from a data structure if it is found.
3. **Searching:** Searching involves searching for the specified data element in a data structure.
4. **Traversal:** Traversal of a data structure means processing all the data elements present in it.
5. **Sorting:** Arranging data elements of a data structure in a specified order is called sorting.
6. **Merging:** Combining elements of two similar data structures to form a new data structure of the same type, is called merging.

- **Algorithm:**

- A well-defined computational procedure that takes some value, or a set of values, as input and produces some value, or a set of values, as output.
- It can also be defined as sequence of computational steps that transform the input into the output.
- An algorithm can be expressed in three ways:-

(i) in any natural language such as English, called pseudo code.

(ii) in a programming language or

(iii) in the form of a flowchart.

- **Efficiency of an Algorithm:**

- In computer science, algorithmic efficiency are the properties of an algorithm which relate to the amount of resources used by the algorithm.
- An algorithm must be analysed to determine its resource usage.
- For maximum efficiency we wish to minimize resource usage.
- However, the various resources (e.g. time, space) can not be compared directly, so which of two algorithms is considered to be more efficient often depends on which measure of efficiency is being considered as the most important

It can be of various types:

1. **Worst case efficiency:**

   It is the maximum number of steps that an algorithm can take for any collection of data values.

2. **Best case efficiency:**

   It is the minimum number of steps that an algorithm can take any collection of data values.

3. **Average case efficiency:**

   It can be defined as - the efficiency averaged on all possible inputs - must assume a distribution of the input

If the input has size n, efficiency will be a function of n

- **Time And Space Complexity (Efficiency):**

Complexity of algorithm is a function of size of input of a given problem instance which determines how much running time/memory space is needed by the algorithm in order to run to completion.

- **Time Complexity:** Time complexity of an algorithm is the amount of time it needs in order to run to completion.
- **Space Complexity:** Space Complexity of an algorithm is the amount of space it needs in order to run to completion.

There are two points which we should consider about computer programming:-

(i) An appropriate data structure

(ii) An appropriate algorithm.

Asymptotic Notations

- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
- For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case.
- But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements i.e. the worst case.
- When the input array is neither sorted nor in reverse order, then it takes average time.
- These durations are denoted using asymptotic notations.

There are mainly three asymptotic notations:

1. Big-O notation
2. Omega notation
3. Theta notation

Big-O Notation

- Big-O notation represents the upper bound of the running time of an algorithm.
- Thus, it gives the worst-case complexity of an algorithm.
- Since it gives the worst-case running time of an algorithm, it is widely used to analyse an algorithm as we are always interested in the worst-case scenario.
- Big O analysis helps program developers to give programmers some basis for computing and measuring the efficiency of a specific algorithm.
- Using Big - O notation, the time taken by the algorithm and the space required to run the algorithm can be ascertained.
- Some of the lists of common computing times of algorithms in order of performance are as follows:

O (1)

O (log n)

O (n)

O (nlog n)

O (n^2) etc.

Thus algorithm with their computational complexity can be rated as per the mentioned order of performance.

- **Array**

- An array is a linear data structure.
- Array is finite collection of similar data items stored in successive or consecutive stored in memory locations.
- For example an array may contains all integer or character elements, but not integer or character elements, but not both.
- Each array can be accessed by using array index and it is must be positive integer value enclosed in square braces.
- This starts from the numerical value 0 and ends at 1 less than size of the array.
- For example an array[n] containing n number of elements are denoted by number of elements are denoted by array[0],array[1],..... array[n-1].
- where '0' is called lower bound and the 'n-1' is called higher bound of an array.

- **Types of Arrays**

Array can be categorized into different types. They are

1. One dimensional array
2. Two dimensional array
3. Multi dimensional array

- **One dimensional array:-**

- One dimensional array is also called as linear array. It is also represents 1-D array.
- the one dimensional array stores the data elements in a single row or column.
- The syntax to declare a linear array is as follows

Syntax:

data type array_name [size];

For example:

int x[6];

Here,
int - type of element to be stored
x - name of the array
6 - size of the array

- **Initialization of one dimensional Array:**

Syntax for the initialization of the linear array is as follows

Syntax:

data type array_name[size]={values};

Example:

int x[6]={2, 4, 7, 20, 18, 11};

- **Memory representation**

One dimensional array:-

a[0]	**a[1]**	**a[2]**	**a[3]**	**a[4]**	**a[5]**	Array
2	**4**	**7**	**20**	**18**	**11**	Array Elements
100	**102**	**104**	**106**	**108**	**110**	Memory addresses

- The memory blocks a[0],a[1],a[2],a[3],a[4] , a[5] with base addresses 100, 102, 104, 106, 108 and 110 store the values 2, 4, 7, 20, 18 and 11 respectively.

- **C++ Array With Empty Members**

➢ In C++, if an array has a size n, we can store up to *n* number of elements in the array.

➢ However, what will happen if we store less than *n* number of elements.

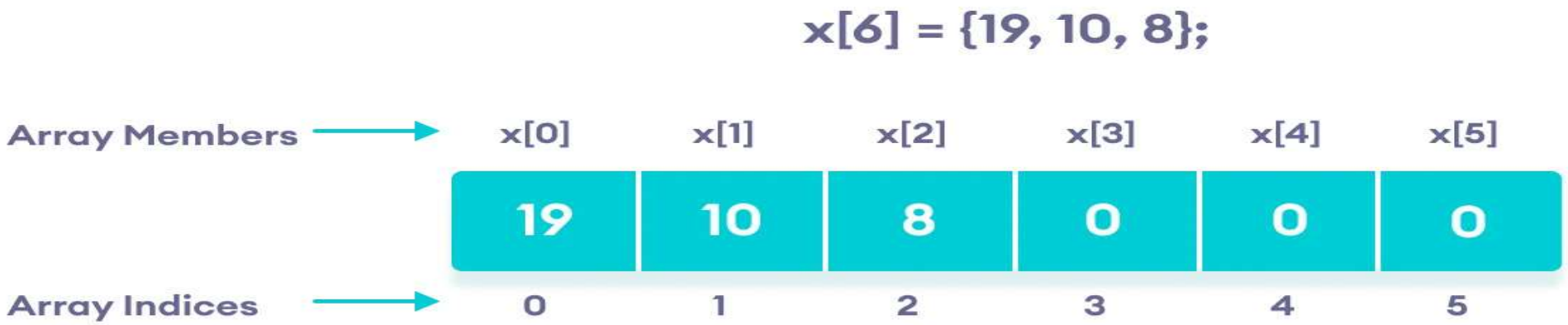
For example,

// store only 3 elements in the array

int x[6] = {19, 10, 8};

➢ Here, the array x has a size of 6. However, we have initialized it with only 3 elements.

➢ In such cases, the compiler assigns random values to the remaining places, this random value is simply 0.



- **Two Dimensional Array (2D Array):**

- Two – dimensional array is the simplest form of a multidimensional array.
- The two-dimensional array can be defined as an array of arrays.
- The 2D array is organized as matrices which can be represented as the collection of rows and columns.
- However, 2D arrays are created to implement a relational database look like data structure.
- It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.
- In 2-D array each element is refer by two indices.
- Elements stored in these Arrays in the form of matrices.
- The first index shows a row of the matrix and the second index shows the column of the matrix.

Syntax of Two-Dimensional Array:-

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

For example:-

```
int X[3] [3];
```

- Here X is 2D array of type integer.
- It contains 3 rows and 3 columns.
- Each element itself is one dimensional array.
- So there will be 3 * 3=9 elements in array X.
- If 2D array X[m][n] is there then this array will contain m * n elements.

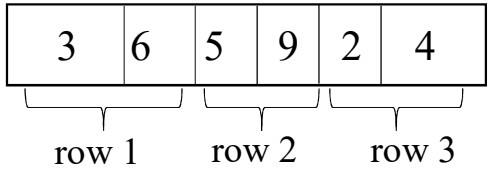
- **Initialization of Two-Dimensional Array:-**

- Here we consider the example of 2D integer array.
- The 2-D arrays which are declared by the keyword int and able to store integer values are called two dimensional integer arrays.
- The process of assigning values during declaration is called initialization.
- These Arrays can be initialized by putting the curly braces around each row separating by a comma also each element of a matrix should be separated by a comma.

Example of a Two Dimensional Integer Array:-

```
int X[3][2] = {3, 6, 5, 9, 2, 4 };
              OR
int X[3][2] = {{ 3,6 }, { 5, 9 }, { 2, 4 } };
```

Array X contains 3 * 2 = 6 elements. So all the 6 elements assign the initial value.



In matrix format elements in X are shown as:

	0	1
0	3	6
1	5	9
2	2	4

- **Sparse Matrix**

- A matrix can be defined as a two-dimensional array having 'm' columns and 'n' rows representing m*n matrix.
- Sparse matrices are those matrices that have the majority of their elements equal to zero.
- In other words, the sparse matrix can be defined as the matrix that has a greater number of zero elements than the non-zero elements.

- **Why do we need to use a sparse matrix instead of a simple matrix?**

- We can also use the simple matrix to store the elements in the memory; then why do we need to use the sparse matrix.
- The following are the advantages of using a sparse matrix:

1. Storage:

- As we know, a sparse matrix that contains lesser non-zero elements than zero so less memory can be used to store elements.
- It evaluates only the non-zero elements.

2. Computing time:

- In the case of searching n sparse matrix, we need to traverse only the non-zero elements rather than traversing all the sparse matrix elements.
- It saves computing time by logically designing a data structure traversing non-zero elements.
- Representing a sparse matrix by a 2D array leads to the wastage of lots of memory.
- The zeroes in the matrix are of no use to store zeroes with non-zero elements.
- To avoid such wastage, we can store only non-zero elements.
- If we store only non-zero elements, it reduces the traversal time and the storage space.

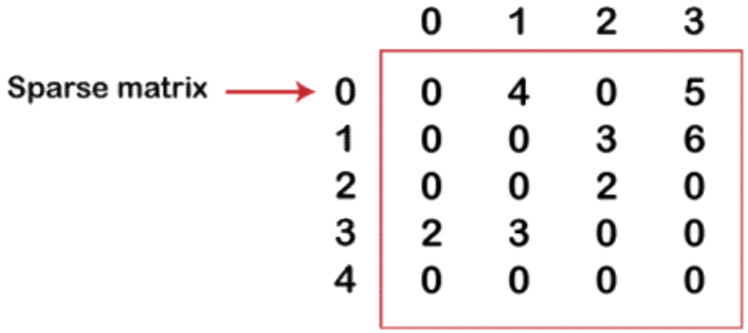
- **Sparse Matrix Representation**

Array Representation

The 2d array can be used to represent a sparse matrix in which there are three rows named as:

1. **Row** : It is an index of a row where a non-zero element is located.
2. **Column** : It is an index of the column where a non-zero element is located.
3. **Value** : The value of the non-zero element is located at the index (row, column).

Let's understand the sparse matrix using array representation through an example.



- As we can observe above, that sparse matrix is represented using triplets, i.e., row, column, and value. In the above sparse matrix, there are 13 zero elements and 7 non-zero elements.
- This sparse matrix occupies 5*4 = 20 memory space.
- If the size of the sparse matrix is increased, then the wastage of memory space will also be increased.

- **Assignments:**

1. Write a C++ program to initialize array and display elements in array.
2. Write a C++ program to input array elements from user and display them using functions (Use input() and display() functions)
3. Write a C++ program to initialize 2D array and display the elements.
4. Write a C++ program to input the elements in 2D array and display them using functions.
5. Write a C++ Menu Driven program to perform following operations on two matrices.
    1. Addition of two matrices
    2. Subtraction of two matrices
    3. Multiplication of two matrices
    4. Transpose of a matrix.

THANK YOU…