Shri G.S. Institute of technology and Science

(An Autonomous Institute, Established in 1952)

LABORATORY ASSIGNMENT

CT10212: DATA STRUCTURE

2024-26

MCA I YEAR

Semester – I

Department of Information Technology

SUBMITED TO:

Mr. Deepesh Agrawal

Ms. Megha Rathore

SUBMITTED BY:

SHIV ARORA

Que 1) What is Data Structure?

Ans 1) A data structure is a way of organizing and storing data in a computer so that it can be accessed and used efficiently. It refers to the logical or mathematical representation of data, as well as the implementation in a computer program.

Characteristics of Data Structures

Data Structure is the systematic way used to organise the data. The characteristics of Data Structures are:

- Linear or Non-Linear

  This characteristic arranges the data in sequential order, such as arrays, graphs etc.

- Static and Dynamic

  Static data structures have fixed formats and sizes along with memory locations. The static characteristic shows the compilation of the data.

- Time Complexity

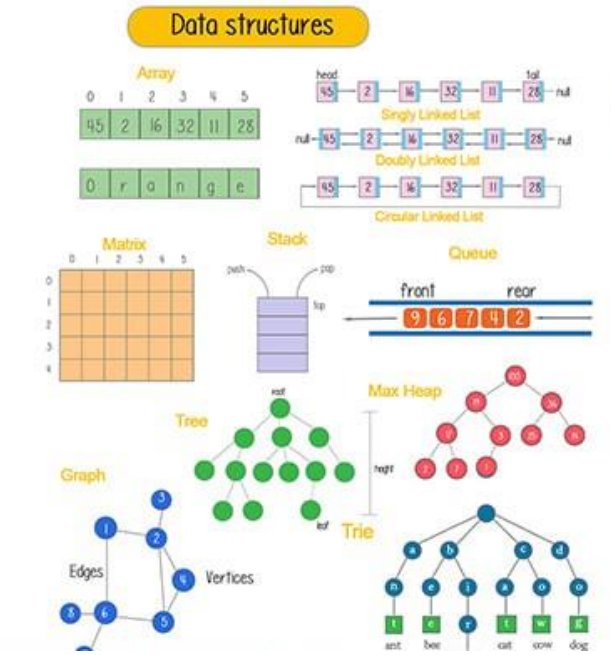
  The time factor should be very punctual. The running time or the execution time of a program should be limited. The running time should be as less as possible. The less the running time, the more accurate the device is.

- Correctness

  Each data must definitely have an interface. Interface depicts the set of data structures. Data Structure should be implemented accurately in the interface.

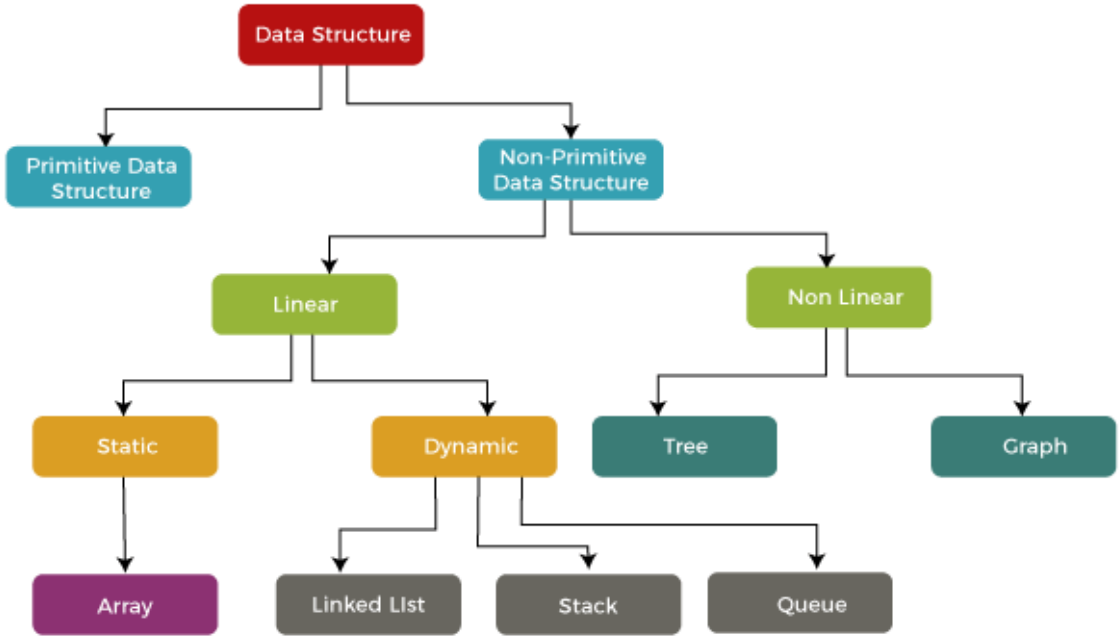
- Space Complexity

  The Space in the device should be managed carefully. The memory usage should be used properly. The space should be less occupied, which indicates the proper function of the device



Que 2) Explain the types of Data Structure?

Ans 2) There are two types of Data Structure Primitive and Non-primitive:



- **Primitive Data Structure:** Primitive data structure is a data structure that can hold a single value in a specific location whereas the non-linear data structure can hold multiple values either in a contiguous location or random locations. The examples of primitive data structure are float, character, integer and pointer. The value to the primitive data structure is provided by the programmer. The following are the four primitive data structures:

a) Integer**:** The integer data type contains the numeric values. It contains the whole numbers that can be either negative or positive. When the range of integer data type is not large enough then in that case, we can use long.

b) Float**:** The float is a data type that can hold decimal values. When the precision of decimal value increases then the Double data type is used.

c) Boolean**:** It is a data type that can hold either a True or a False value. It is mainly used for checking the condition.

d) Character**:** It is a data type that can hold a single character value both uppercase and lowercase such as 'A' or 'a'.

- **Non-primitive Data Structure:** The non-primitive data structure is a kind of data structure that can hold multiple values either in a contiguous or random location. The non-primitive data types are defined by the programmer. The non-primitive data structure is further classified into two categories, i.e., linear and non-linear data structure. In case of linear data structure, the data is stored in a sequence, i.e., one data after another data. When we access the data from the linear data structure, we just need to start from one place and will find other data in a sequence.

- **Linear Data Structure:** A data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure. Examples are array, stack, queue, etc.

a) Array**:** An array is a data structure that can hold the elements of same type. It cannot contain the elements of different types like integer with character. The commonly used operation in an array is insertion, deletion, traversing, searching.

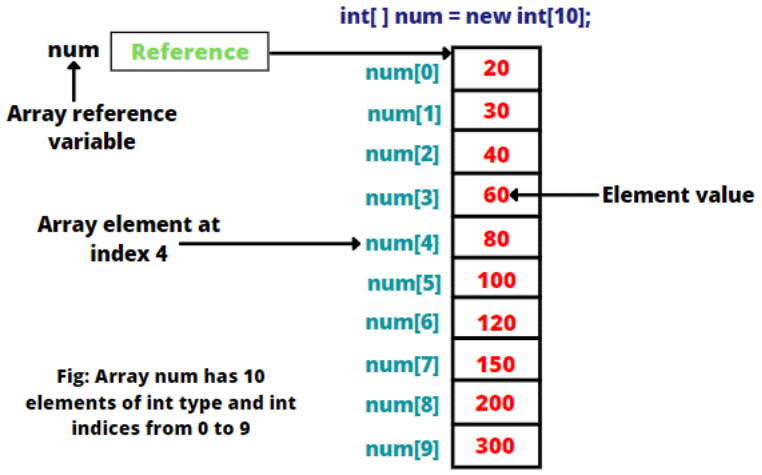
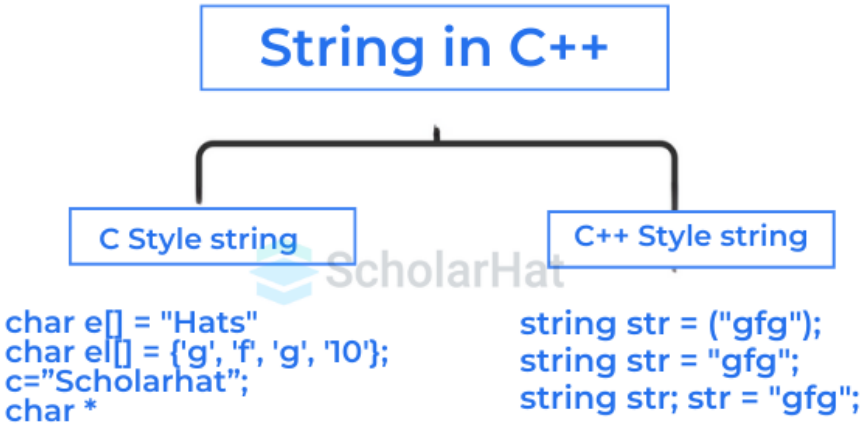
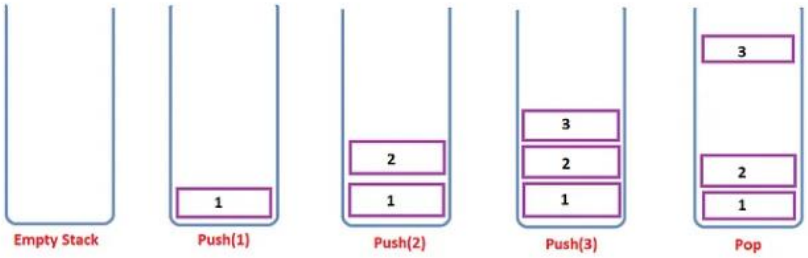


Fig: Array num has 10 elements of int type and int indices from 0 to 9

b) String**:** String is defined as an array of characters. The difference between the character array and string is that the string data structure terminates with a 'NULL' character, and it is denoted as a '\0'.

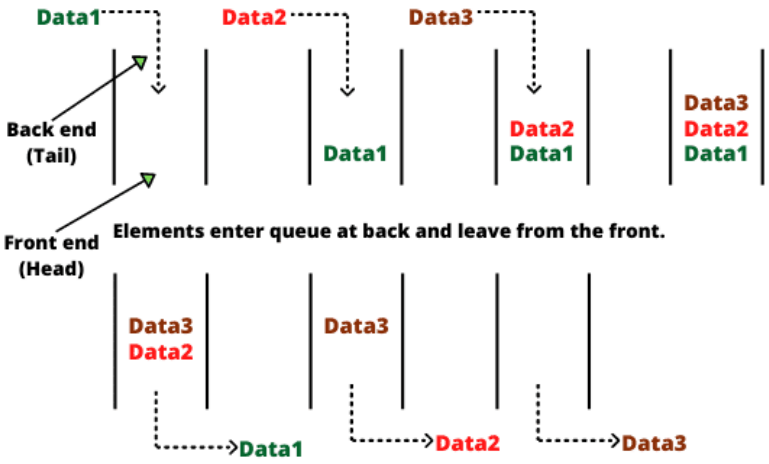


c) Stack**:** Stack is a data structure that follows the principal **LIFO** (Last in First Out). All the operations on the stack are performed from the top of the stack such as PUSH and POP operation. The push operation is the process of inserting element into the stack while the pop operation is the process of removing element from the stack. The stack data structure can be implemented by using either array or linked list.



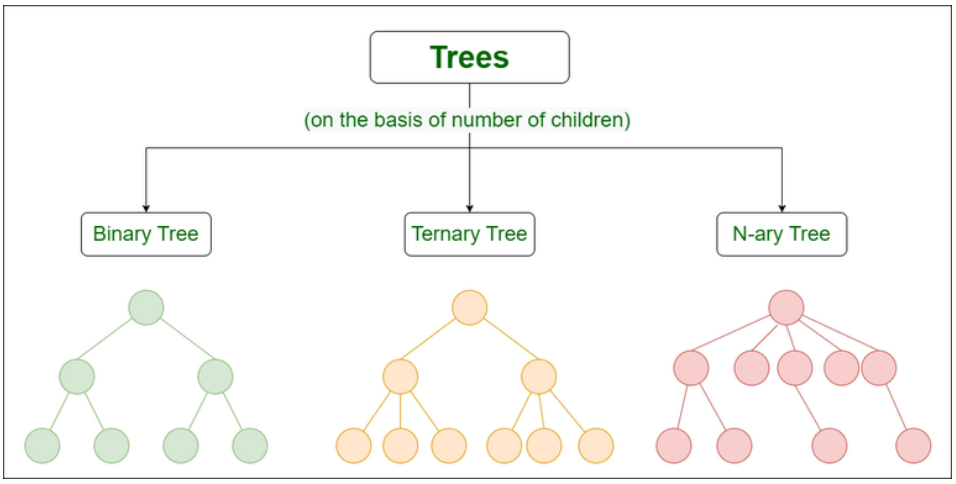
Stack Data Structure

d) Queue**:** Queue is a data structure that can be implemented by using array. The difference between the stack and queue data structure is that the elements in the queue are inserted from the rear end while the elements in the queue are removed from the front end.



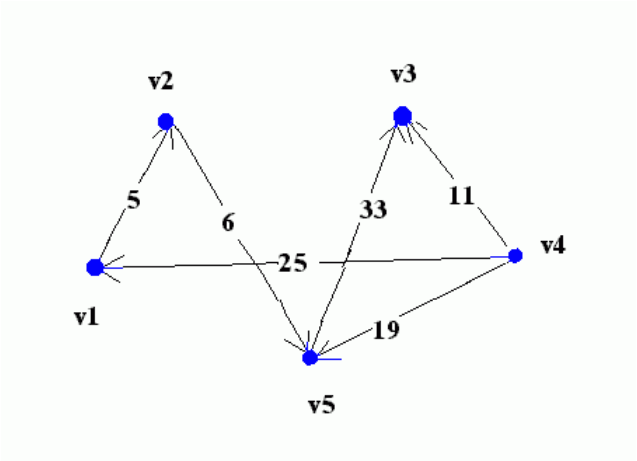
- **Non-linear Data Structure:** Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. Examples are trees and graphs.

a) Tree**:** A tree data structure is a hierarchical structure consisting of nodes connected by edges, where each tree has a single root node. Nodes can have zero or more child nodes, forming parent-child relationships. Leaf nodes are those without any children, while internal nodes have at least one child. The height of a tree is defined by the longest path from the root to a leaf, whereas the depth measures the distance from the root to a specific node. Trees facilitate efficient data organization and retrieval, making them essential for various applications like databases, file systems, and algorithms. Overall, they provide a structured way to represent relationships and enable quick access to information.



b) Graph**:** A graph data structure is a collection of nodes, or vertices, connected by edges that represent relationships between them. Unlike trees, graphs can have cycles and do not require a hierarchical structure, allowing for more complex relationships. Each edge can be directed or undirected, indicating the nature of the relationship. Graphs can also be weighted, where edges carry values representing costs, distances, or capacities. They are widely used in computer science for applications such as social networks, transportation

systems, and network topology. Overall, graphs provide a flexible framework for modelling a variety of interconnected data.



Que 3) What are the Difference between Linear and Non- Linear Data Structure?

Ans 3) Here are some differences between linear and non-linear data structures:

1. **Definition**:
   - Linear: Elements are arranged sequentially.
   - Non-Linear: Elements are arranged hierarchically or in a network.
2. **Memory Allocation**:
   - Linear: Typically requires contiguous memory allocation.
   - Non-Linear: Can use non-contiguous memory allocation.
3. **Structure**:
   - Linear: Form a straight line (e.g., arrays, linked lists).
   - Non-Linear: Form complex structures (e.g., trees, graphs).
4. **Data Access**:
   - Linear: Access is sequential.
   - Non-Linear: Access can be more complex and may require traversals.
5. **Traversal Methods**:
   - Linear: Traversal is straightforward (e.g., from start to end).
   - Non-Linear: Traversal can be multiple ways (e.g., depth-first, breadth-first).
6. **Examples**:
   - Linear: Arrays, linked lists, stacks, queues.
   - Non-Linear: Trees, graphs, heaps.
7. **Use Cases**:
   - Linear: Best for simple data storage and retrieval.
   - Non-Linear: Best for representing hierarchical data or complex relationships.
8. **Data Representation**:
   - Linear: Data elements have a unique successor and predecessor.
   - Non-Linear: Data elements can have multiple predecessors and successors.

9. **Complexity**:
    - Linear: Generally simpler in implementation and understanding.
    - Non-Linear: More complex and requires advanced algorithms for manipulation.
10. **Performance**:
    - Linear: Typically, faster for searching and accessing elements due to sequential layout.
    - Non-Linear: Performance can vary greatly based on structure and algorithms used.
11. **Scalability**:
    - Linear: Less scalable due to fixed size (especially arrays).
    - Non-Linear: More scalable; can grow dynamically (e.g., trees).
12. **Data Relationships**:
    - Linear: Relationships are one-to-one (linear).
    - Non-Linear: Relationships can be one-to-many or many-to-many.
13. **Insertion/Deletion**:
    - Linear: Generally, takes O(n) time in arrays; faster in linked lists.
    - Non-Linear: Can be more efficient, especially in trees (e.g., O (log n) in balanced trees).
14. **Examples of Operations**:
    - Linear: Push/pop in stacks, enqueue/dequeue in queues.
    - Non-Linear: Searching paths in graphs, inserting nodes in trees.
15. **Applications**:
    - Linear: Used in simple data management, such as lists and queues.
    - Non-Linear: Used in databases, AI (graphs), and organizational structures.
16. **Implementation Complexity**:
    - Linear: Easier to implement due to straightforward operations.
    - Non-Linear: More challenging due to the need for pointers and managing relationships.
17. **Sorting**:
    - Linear: Sorting can be done easily (e.g., arrays).
    - Non-Linear: Sorting is more complex (e.g., trees have specific orders).
18. **Data Access Patterns**:
    - Linear: Access patterns are predictable.
    - Non-Linear: Access patterns can be irregular and varied.
19. **Example Algorithms**:
    - Linear: Algorithms like bubble sort or linear search.
    - Non-Linear: Algorithms like Dijkstra's for shortest paths or tree traversals.

Que 4) Difference between Data and information?

Ans 4) Data and information are fundamental concepts in the fields of computing, data science, and information theory. While they are often used interchangeably in everyday conversation, they have distinct meanings and roles.

Data

Data refers to raw facts and figures without context. It can be quantitative (numbers) or qualitative (descriptive) and can exist in various forms, such as text, numbers, images, or audio.

Characteristics:

- Raw and Unprocessed**:** Data is unrefined and lacks context, meaning, or relevance.
- Discrete**:** Data points can exist independently and may not tell a complete story.
- Quantifiable**:** Data can often be measured and represented numerically, though qualitative data can also be considered.
- Types**:** Data can be structured (like data in a database), semi-structured (like JSON or XML files), or unstructured (like text documents or images).

Examples:

- A list of temperatures recorded every hour.
- Individual customer records in a database (name, age, purchase history).
- Survey responses (e.g., "Yes," "No," or "Maybe").

Information

Information is processed, organized, or structured data that has meaning and context. It provides answers to questions and adds value to data by making it understandable.

Characteristics:

- Contextualized**:** Information is data that has been interpreted and put into context, making it useful for decision-making.
- Relevant**:** Information is often specific to a particular situation or problem, providing insight that data alone cannot.
- Processed**:** Information results from analysing, organizing, and summarizing data.
- Communicative**:** Information is typically conveyed in a way that is easily understood by its intended audience.

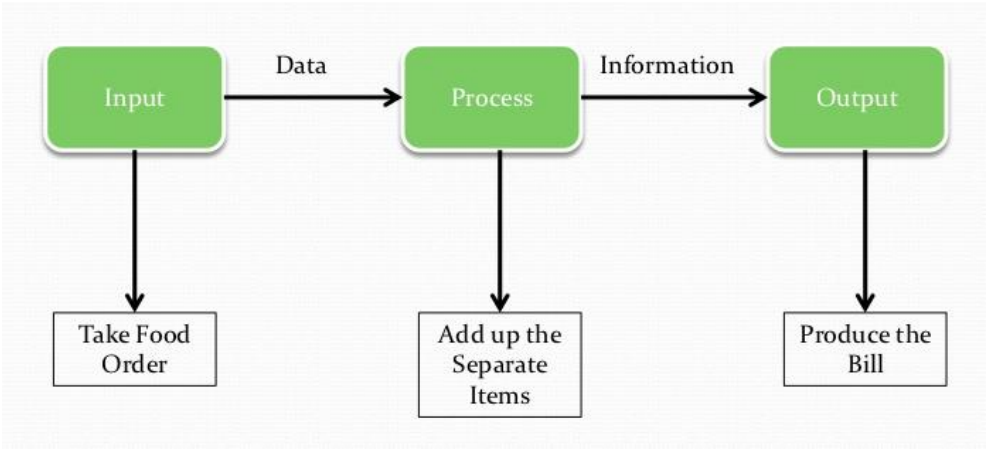
Examples:

- A weather report summarizing the temperature trends and predicting future weather conditions based on hourly data.
- A customer profile that combines various data points to provide insights about purchasing behaviour.
- A research paper that interprets survey data to draw conclusions about public opinion.

Relationship Between Data and Information

- Transformation**:** Data becomes information when it is processed and given context. This can involve sorting, filtering, analysing, and interpreting data.

- Hierarchy**:** Data serves as the building blocks for information. Information is derived from data and depends on the quality and relevance of the data used.
- Usage**:** In decision-making, stakeholders rely on information derived from data. Good data leads to reliable information, which in turn supports effective decision-making.



Que 5) What is Abstract Data Types? Explain ADT stack, Queue and linked list?

Ans 5) An Abstract Data Type (ADT) is a mathematical model for certain data types that defines their behaviour from the point of view of a user, specifically the operations that can be performed on them and the properties of those operations. ADTs focus on what operations are to be performed rather than how they are implemented. This abstraction allows for better flexibility and modular design in programming.

Stack

A stack is an ADT that operates on a Last In, First Out (LIFO) principle. This means that the last item added to the stack is the first one to be removed.

Basic Operations:

- Push(): Add an item to the top of the stack.
- Pop()**:** Remove the item from the top of the stack.
- Peek/Top(): Retrieve the item at the top of the stack without removing it.
- IsEmpty(): Check if the stack is empty.

Example Use Cases:

- Undo mechanisms in text editors.
- Function call management in programming languages (call stack).
- Expression evaluation and syntax parsing.

Queue

A queue is an ADT that operates on a First In, First Out (FIFO) principle. This means that the first item added to the queue is the first one to be removed.

Basic Operations:

- Enqueue()**:** Add an item to the rear of the queue.
- Dequeue()**:** Remove the item from the front of the queue.
- Front/Peek()**:** Retrieve the item at the front of the queue without removing it.
- IsEmpty()**:** Check if the queue is empty.

Example Use Cases:

- Print job management in printers.
- Task scheduling in operating systems.
- Breadth-First Search (BFS) in graph algorithms.

Linked List

A linked list is an ADT that consists of a sequence of elements, where each element (node) contains a value and a reference (link) to the next node in the sequence. Linked lists can be singly linked (each node points to the next one) or doubly linked (each node points to both the next and previous nodes).

Basic Operations:

- Insert()**:** Add a node at a specified position (beginning, end, or specific index).
- Delete()**:** Remove a node from a specified position.
- Search()**:** Find a node with a specific value.
- Traverse()**:** Visit each node in the list.

Example Use Cases:

- Dynamic memory allocation.
- Implementing stacks and queues.
- Managing lists of items (like a playlist).