Program Core Course - Data Structures

Question Bank Solution - IAT 1 by Syntax Syndicate

Reference - (Mentioned in Syllabus)

1. Sahni Horowitz, Fundamentals of data structures in C, computer science press, 2008.
2. Jean Paul Tremblay, P. G. Sorenson, “Introduction to Data Structure and Its Applications”, McGraw-Hill Higher Education

---

1) Define Data Structure.

Ans.:

A **Data Structure** is a systematic way of organizing, storing and managing data in memory so that it can be accessed and modified efficiently.

1. It specifies how data elements are arranged in memory.
2. The relationship between data items is clearly defined.
3. Various operations such as insertion, deletion, searching, and traversal can be performed efficiently.
4. Proper organization of data improves program performance.
5. Selection of an appropriate data structure reduces time and space complexity.
6. Data structures are classified into:
    - Linear Data Structures
    - Non-Linear Data Structures
7. Examples include Array, Stack, Queue, Linked List, and Tree.

---

2. What is the need for Data Structures?

Ans.:

Data structures are required to organize data efficiently and to solve computational problems effectively.

1. When the size of data increases, simple variables are not sufficient to manage the data efficiently.
2. Proper organization of data allows faster access and modification.
3. Efficient searching and sorting operations depend on appropriate data structures.
4. Memory utilization becomes better when dynamic structures allocate memory as required.
5. Many algorithms are designed based on specific data structures such as stacks, queues, and trees.
6. Real-world applications such as operating systems, databases, and compilers rely on efficient data handling.

7. For example, stack is used in recursion and expression evaluation, queue is used in CPU scheduling, and trees are used in file directory systems.
8. Therefore, data structures improve overall performance and efficiency of programs.

---

3. State the difference between Data Type and Data Structure.

Ans.:

Basis of Comparison	Data Type	Data Structure
Definition	A data type defines the type of value that a variable can store.	A data structure defines the way in which multiple data elements are organized and arranged in memory.
Purpose	Used to specify the kind of data such as integer, float, or character.	Used to organize and manage a collection of data efficiently.
Nature	Basic and fundamental concept of programming.	Advanced concept built using data types.
Memory Allocation	Memory size is fixed and predefined according to the type.	Memory may be static (array) or dynamic (linked list).
Data Handling	Handles single data values.	Handles a group or collection of data elements.
Complexity	Simple and primitive in nature.	More complex and structured in nature.
Examples	int, float, char, double	Array, Stack, Queue, Linked List, Tree

4. Give examples of real-life data structures.

Ans.:

Data structures can be easily understood by comparing them with real-life systems.

1. A stack can be compared to a stack of plates where the last plate placed on top is removed first, following LIFO principle.
2. A queue is similar to people standing in a line at a ticket counter where the person who comes first is served first, following FIFO principle.
3. A linked list can be compared to train coaches where each coach is connected to the next one.
4. A tree structure resembles a family tree or organizational hierarchy where one root node branches into multiple child nodes.
5. A graph can be compared to a social network where users are connected through relationships.
6. An array can be compared to students sitting in a row where each student has a fixed position.
7. These real-life examples help in understanding abstract data structures more clearly.

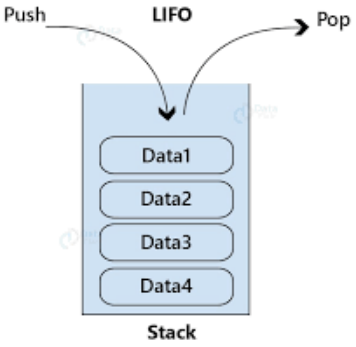
---

5. Define Stack.

Ans.:

A Stack is a linear data structure in which insertion and deletion operations are performed only at one end called TOP, and it follows the Last In First Out (LIFO) principle.

1. Stack allows operations only at one end, unlike arrays where access is allowed at any position.
2. The element inserted last is the first element to be removed.
3. The insertion operation in stack is called PUSH.
4. The deletion operation in stack is called POP.
5. A variable called TOP is used to keep track of the last inserted element.
6. Stack can be implemented using arrays (static implementation) or linked lists (dynamic implementation).
7. Stack is used in recursion, expression evaluation, parentheses checking, and reversing strings.



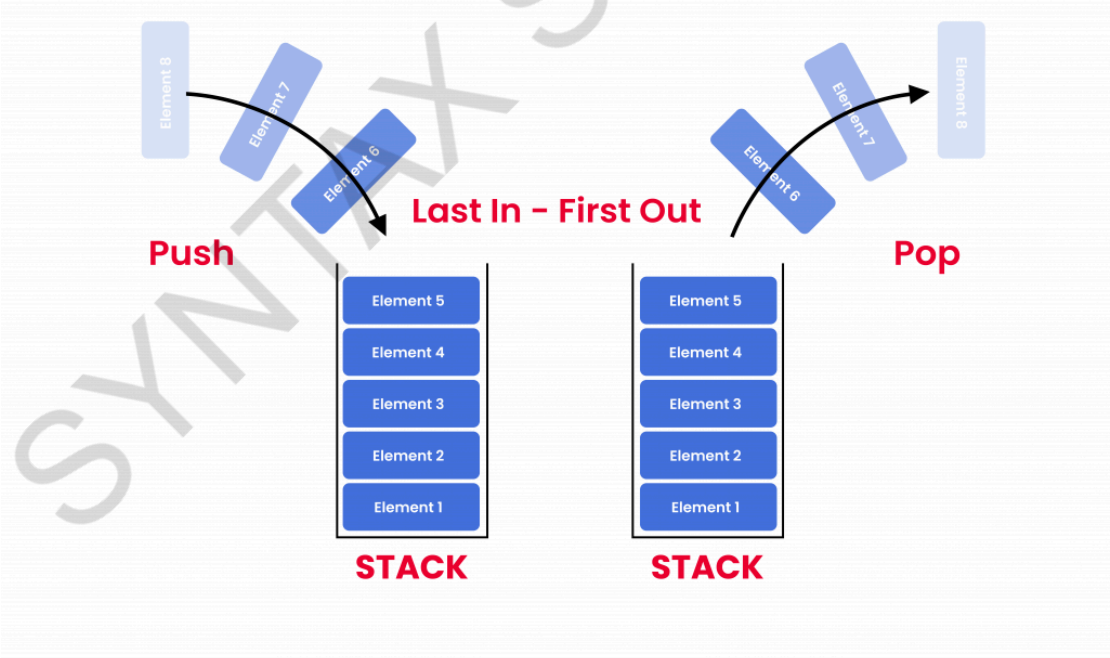
---

6. What is LIFO Principle?

Ans.:

LIFO stands for **Last In First Out**. It is the principle followed by the stack data structure.

1. In LIFO, the element inserted last is removed first.
2. All insertion and deletion operations are performed at only one end called **TOP**.
3. The most recently added element is the first one to be deleted.
4. This principle restricts access to only one element at a time (the top element).
5. LIFO is the fundamental rule that defines the behavior of a stack.



---

7. State applications of stack (any two)

Ans.:

Stack is widely used in many real-life and programming applications.

1. **Expression Evaluation**
   Stack is used to evaluate postfix and prefix expressions efficiently.
2. **Conversion of Expressions**
   Infix expressions are converted to postfix or prefix using stack.
3. **Parentheses Checking**
   Stack is used to check whether parentheses in an expression are balanced.
4. **Function Calls and Recursion**
   During recursion, function calls are stored in the call stack.
5. **Reversing a String or List**
   Elements are pushed into stack and popped to reverse their order.

---

8. What are Push and Pop operations?

Ans.:

Push and Pop are the two fundamental operations performed on a stack.

Push Operation:

Push is the operation of inserting an element into the stack.

1. Before insertion, overflow condition is checked.
2. If stack is not full, TOP is incremented.
3. The new element is inserted at the TOP position.
4. Time complexity of push operation is O(1).

Pop Operation:

Pop is the operation of removing an element from the stack.

1. Before deletion, underflow condition is checked.
2. The element at TOP is removed.
3. TOP is decremented after deletion.
4. Time complexity of pop operation is O(1).

---

9. Define Overflow and Underflow in Stack.

Ans.:

Overflow and Underflow are error conditions in stack operations.

Overflow:

Overflow occurs when an attempt is made to push an element into a stack that is already full.

1. Happens in array implementation when TOP = MAX - 1.
2. No more elements can be inserted.
3. Indicates stack memory is full.
4. Further push operations are not allowed.

Underflow:

Underflow occurs when an attempt is made to pop an element from an empty stack.

1. Happens when TOP = -1.
2. No element is available to delete.
3. Indicates stack is empty.
4. Further pop operations are not possible.

---

10. Write a program to implement stack using an array.

Ans.:

```
#include <stdio.h>

#define MAX 5

int stack[MAX];

int top = -1;

void push()

{

   int x;

   if(top == MAX-1)

      printf("Stack Overflow\n");

   else

   {

      printf("Enter element: ");
```

```
        scanf("%d", &x);
        top++;
        stack[top] = x;
    }
}
void pop()
{
    if(top == -1)
        printf("Stack Underflow\n");
    else
    {
        printf("Deleted element: %d\n", stack[top]);
        top--;
    }
}
void display()
{
    int i;
    if(top == -1)
        printf("Stack is Empty\n");
    else
    {
        printf("Stack elements:\n");
        for(i = top; i >= 0; i--)
            printf("%d\n", stack[i]);
    }
}
int main()
```

```
{
    int ch;
    do
    {
        printf("\n1.Push 2.Pop 3.Display 4.Exit\n");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: display(); break;
        }
    }while(ch != 4);
    return 0;
}
```

Algorithms (Optional) :

Algorithm for PUSH

1. Start
2. Check if `top == MAX - 1`
3. If true, print "Stack Overflow" and stop
4. Otherwise, increment `top`
5. Insert element at `stack[top]`
6. Stop

Algorithm for POP

1. Start
2. Check if `top == -1`
3. If true, print "Stack Underflow" and stop
4. Otherwise, print `stack[top]`
5. Decrement `top`

6. Stop

Algorithm for DISPLAY

1. Start
2. Check if `top == -1`
3. If true, print "Stack is Empty"
4. Otherwise, print elements from `top` to `0`
5. Stop

---

11. Define Double Ended Queue (Deque)

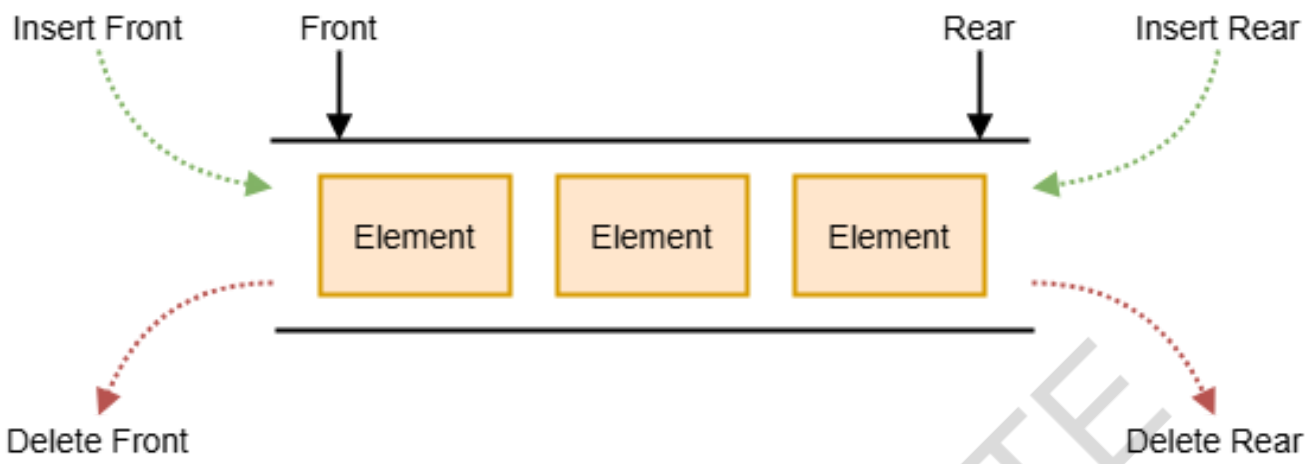
Ans.:

A Double Ended Queue (Deque) is a linear data structure in which insertion and deletion of elements can be performed at both ends, namely front and rear.

1. Unlike a simple queue, deque allows operations at both ends.
2. Elements can be inserted either at the front or at the rear.
3. Elements can also be deleted from both front and rear.
4. Deque does not strictly follow FIFO principle.
5. It provides more flexibility compared to a normal queue.
6. Deque can be implemented using arrays or linked lists.

There are two types of Deque:

- **Input Restricted Deque** – Insertion allowed at one end but deletion allowed at both ends.
- **Output Restricted Deque** – Deletion allowed at one end but insertion allowed at both ends.



---

12. What is the difference between Linear Queue and Circular Queue?

Ans.:

Basis	Linear Queue	Circular Queue
Structure	Elements are arranged in a straight sequence.	Last position is connected back to the first position forming a circular structure.
Memory Utilization	Wastage of memory may occur when front moves forward.	Memory is efficiently utilized as empty spaces are reused.
Movement of Rear	Rear pointer moves only in forward direction.	Rear pointer moves circularly using modulo operation.
Condition for Full	`rear == MAX - 1`	`(rear + 1) % MAX == front`

Space Reuse	Freed spaces at beginning cannot be reused.	Freed spaces are reused automatically.
Need for Shifting	Sometimes shifting of elements is required.	No shifting of elements is required.

7-Mark Questions

13. Define Abstract Data Type (ADT) and explain its features with examples

Ans.:

An Abstract Data Type (ADT) is a logical model of a data structure that defines the data and the operations that can be performed on it without specifying the implementation details.

1. ADT focuses on what operations are to be performed rather than how they are implemented.
2. It provides separation between interface and implementation.
3. Internal details are hidden from the user (data hiding).
4. Improves modularity and security of programs.
5. Same ADT can have multiple implementations.

Example:

Stack ADT defines operations:

- push()
- pop()
- peek()
- isEmpty()

It does not specify whether stack is implemented using array or linked list.

Queue ADT defines:

- enqueue()
- dequeue()
- isFull()
- isEmpty()

Thus, ADT provides a theoretical framework for data structures.

14. Convert the following infix expression to postfix using stack

((A+B)*C-(D-E))$(F+G)

Ans . :

Step-by-Step Conversion

((A+B)*C-(D-E))$(F+G)

→ ((AB+)*C-(D-E))$(F+G)

→ (AB+C*-(D-E))$(F+G)

→ (AB+C*-(DE-))$(F+G)

→ (AB+C*DE--)$(F+G)

→ AB+C*DE--$(F+G)

→ AB+C*DE--$(FG+)

→ AB+C*DE--FG+$

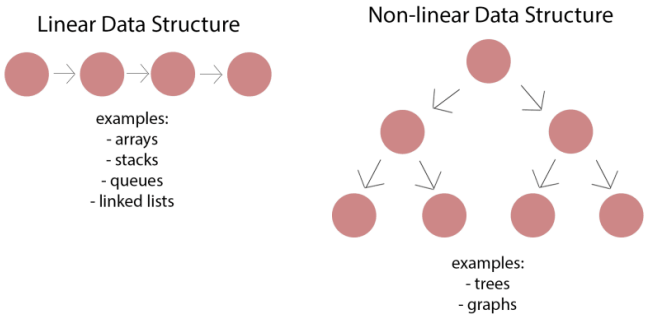
Final Postfix Expression

`AB+C*DE--FG+$`

---

15. Explain Linear vs Non-Linear Data Structures with suitable real-world applications.

Ans.:



Linear Data Structure

A linear data structure is one in which elements are arranged sequentially, and each element (except the first and last) has exactly one predecessor and one successor.

1. Data elements are stored in a single continuous level.
2. Each element is connected to its immediate previous and next element.
3. Traversal is performed sequentially from beginning to end.
4. Implementation is comparatively simple.
5. Memory allocation may be contiguous (Array) or non-contiguous (Linked List).
6. Examples include Array, Stack, Queue, and Linked List.

Real-Life Examples of Linear Data Structures

1. Stack – Stack of Plates
   Plates are placed one over another. The last plate placed is removed first (LIFO principle).
   Application: Undo operation in text editors.

2. Queue – People Standing in Line
   The person who comes first is served first (FIFO principle).
   Application: Printer queue, ticket counter.

3. Array – Students Sitting in a Row
   Each student has a fixed position number.
   Application: Storing marks of students.

4. Linked List – Train Coaches
   Each coach is connected to the next coach.
   Application: Music playlist navigation.

Non-Linear Data Structure

A non-linear data structure is one in which elements are arranged hierarchically or interconnected, and one element can be connected to multiple elements.

1. Data elements are not arranged sequentially.
2. A single element may have multiple predecessors or successors.
3. Traversal cannot be done in a simple sequential manner.
4. Used to represent hierarchical or network relationships.
5. Examples include Tree and Graph.

Real-Life Examples of Non-Linear Data Structures

1. Tree – File Directory System
   A root folder contains subfolders, and each subfolder may contain more files or folders.

Application: Windows file system.

2. Graph – Social Network
   Users are connected to many other users.
   Application: Facebook friend connections, Instagram followers.

3. Organization Hierarchy
   CEO at top, managers below, employees under managers.
   Application: Company structure representation.

---

16. Explain Infix, Postfix and Prefix Expressions with an Example.

Ans.:

An expression is a combination of operands and operators used to perform calculations.

Infix Expression

In infix expression, the operator is written between operands.

Example:

(A + B) * C

1. Operator is placed between operands.
2. Most common form used in mathematics.
3. Requires parentheses to maintain precedence.
4. Evaluation needs operator precedence rules.

Postfix Expression

In postfix expression, the operator is written after the operands.

Example:

AB+

1. Operator comes after operands.
2. Does not require parentheses.
3. Easy to evaluate using stack.
4. Scanned from left to right.

Example Conversion:

Infix: (A + B) * C

Postfix: AB+C*

Prefix Expression

In prefix expression, the operator is written before the operands.

Example:

+AB

1. Operator comes before operands.
2. Does not require parentheses.
3. Evaluated using stack.
4. Scanned from right to left.

Example Conversion:

Infix: (A + B) * C

Prefix: *+ABC

---

17. Compare stack and queues.

Ans.:

Basis	Stack	Queue
Principle	Follows LIFO (Last In First Out)	Follows FIFO (First In First Out)
Insertion	Insertion is done at TOP (Push)	Insertion is done at REAR (Enqueue)
Deletion	Deletion is done from TOP (Pop)	Deletion is done from FRONT (Dequeue)
Ends Used	Only one end is used for both insertion and deletion	Two ends are used (Front and Rear)

Order of Removal	Last inserted element is removed first	First inserted element is removed first
Implementation	Can be implemented using array or linked list	Can be implemented using array or linked list
Applications	Expression evaluation, recursion, undo operations	CPU scheduling, printer queue, buffering

18. Explain the Different Types of Queues

A queue is a linear data structure that follows FIFO principle.

1. Linear Queue

1. Elements inserted at rear and deleted from front.
2. Rear moves only forward.
3. May cause memory wastage.

Application: Ticket counter system.

2. Circular Queue

1. Last position connects back to first.
2. Rear moves in circular manner.
3. Efficient memory utilization.
4. Condition for full:
   (Rear + 1) % MAX = Front

Application: CPU scheduling.

3. Priority Queue

1. Each element has a priority.
2. Element with highest priority is deleted first.
3. Does not strictly follow FIFO.

Application: Emergency patient treatment system.

4. Double Ended Queue (Deque)

1. Insertion and deletion allowed at both ends.
2. Can act as stack or queue.

Application: Browser history.

---

19. Explain the Working of Circular Queue with Example

Ans.:

A circular queue is a queue in which the last position is connected back to the first position to form a circle.

1. Two pointers are used: Front and Rear.
2. Initially, Front = Rear = -1.
3. First insertion: Front = Rear = 0.
4. Rear moves using formula:
   Rear = (Rear + 1) % MAX
5. Deletion moves Front similarly.
6. Queue is full when:
   (Rear + 1) % MAX = Front

Insert 10, 20, 30, 40

After inserting 10

```
F=0  R=0

[10] [ ] [ ] [ ]
```

After inserting 20

```
F=0  R=1

[10] [20] [ ] [ ]
```

After inserting 30

```
F=0  R=2

[10] [20] [30] [ ]
```

After inserting 40

```
F=0  R=3

[10] [20] [30] [40]
```

Queue is now full.

---

20. Write Program in C to Evaluate a Postfix Expression Using Stack ADT

Ans.:

Algorithm

1. Create an empty stack.
2. Scan postfix expression from left to right.
3. If operand, push onto stack.
4. If operator, pop two operands.
5. Perform operation.
6. Push result back to stack.
7. After scanning complete, top element is result.

C Program

```
#include <stdio.h>

#include <ctype.h>

#include <math.h>

#define MAX 100

int stack[MAX];

int top = -1;

void push(int x) {

   stack[++top] = x;

}
```

```
int pop() {
    return stack[top--];
}
int main() {
    char exp[100];
    int i, op1, op2, result;
    printf("Enter postfix expression: ");
    scanf("%s", exp);
    for(i = 0; exp[i] != '\0'; i++) {
        if(isdigit(exp[i])) {
            push(exp[i] - '0');
        }
        else {
            op2 = pop();
            op1 = pop();
            switch(exp[i]) {
                case '+': result = op1 + op2; break;
                case '-': result = op1 - op2; break;
                case '*': result = op1 * op2; break;
                case '/': result = op1 / op2; break;
                case '$': result = pow(op1, op2); break;
            }
            push(result);
        }
    }
    printf("Result = %d", pop());
    return 0;
}
```

Questions From Another Question Bank

1. Explain Linear Data Structures and discuss Arrays, Linked Lists, Stacks, and Queues in brief.

Ans.:

Linear Data Structure

A linear data structure is a structure in which elements are arranged sequentially, one after another. Each element has a unique predecessor and successor except the first and last elements.

- Elements are stored in a single level.
- Traversal is done sequentially.
- Memory may be contiguous or non-contiguous.
- Easier to implement compared to non-linear structures.

(a) Array

An array is a collection of elements of the same data type stored in contiguous memory locations.

- Elements are accessed using index.
- Random access is possible.
- Size is fixed.
- Insertion and deletion are costly.

Diagram:

```
Index:  0   1   2   3

       [10][20][30][40]
```

Applications:
- Storing marks of students
- Matrix representation
- Lookup tables

(b) Linked List

A linked list is a linear data structure where elements (nodes) are connected using pointers.

• Memory is non-contiguous.
• Dynamic size.
• Easy insertion and deletion.
• Requires extra memory for pointer.

Diagram:

```
[10 | *] → [20 | *] → [30 | NULL]
```

Applications:
• Dynamic memory management
• Implementation of stack and queue
• Polynomial representation

(c) Stack

A stack is a linear data structure that follows LIFO (Last In First Out).

• Insertion operation: Push
• Deletion operation: Pop
• Only one end called TOP is used.

Diagram:

```
  TOP

   [30]

   [20]

   [10]
```

Applications:
• Expression evaluation
• Function calls (recursion)
• Undo/Redo operations

(d) Queue

A queue is a linear data structure that follows FIFO (First In First Out).

• Insertion at Rear (Enqueue)
• Deletion at Front (Dequeue)
• Two pointers: Front and Rear

Diagram:

```
Front → [10][20][30] ← Rear
```

Applications:
- CPU scheduling
- Printer queue
- Buffering in networking

---

2. Compare Sequential and Linked Representation of Linear Data Structures

Ans.:

Basis	Sequential Representation	Linked Representation
Memory Allocation	Memory is allocated in contiguous locations.	Memory is allocated in non-contiguous locations.
Implementation	Implemented using arrays.	Implemented using linked lists (nodes with pointers).
Structure	Elements are stored one after another in sequence.	Elements are stored as nodes connected through pointers.
Size	Size is fixed at the time of declaration.	Size is dynamic and can grow or shrink during execution.
Insertion	Insertion is costly because shifting of elements is required.	Insertion is easier; only pointer modification is required.
Deletion	Deletion is costly due to shifting of elements.	Deletion is easier; pointer links are adjusted.

Memory Utilization	May cause memory wastage if declared size is not fully used.	Efficient memory utilization; memory allocated as needed.
Access Time	Direct access possible using index (O(1)).	Only sequential access possible (O(n)).
Overflow Condition	Occurs when array becomes full.	Occurs only when memory is exhausted.
Extra Memory	No extra memory required for pointers.	Extra memory required to store pointer in each node.

3. Differentiate array and linked list with advantages, disadvantages, and applications.

Ans.:

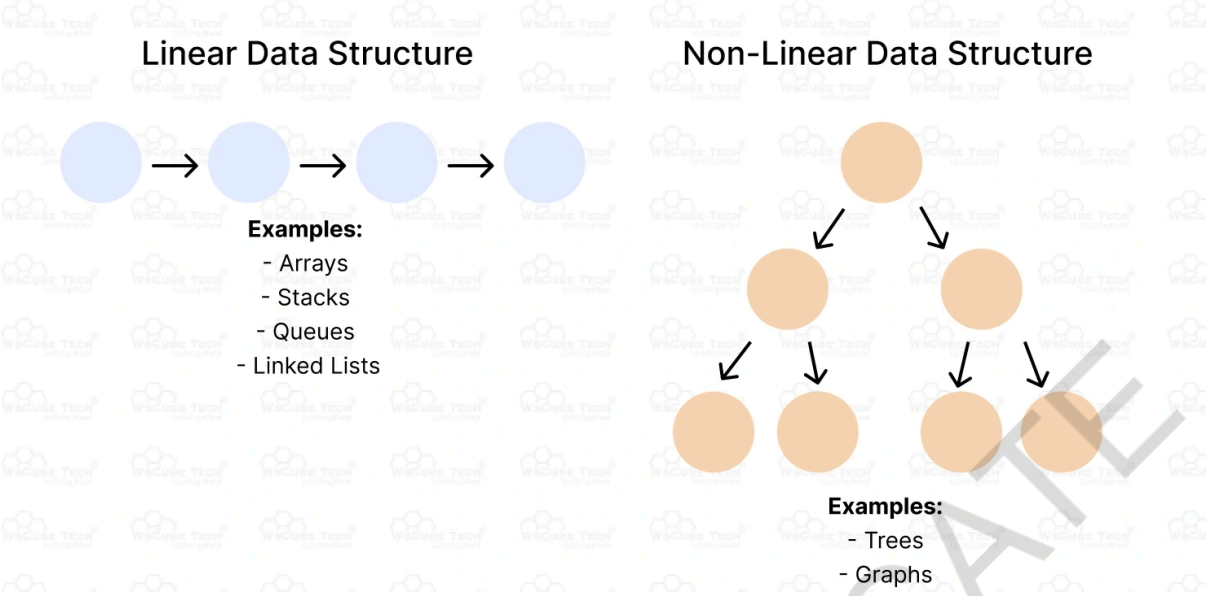
Basis	Array	Linked List
Definition	Array is a linear data structure that stores elements of the same data type in contiguous memory locations.	Linked list is a linear data structure in which elements (nodes) are stored in non-contiguous memory locations and connected using pointers.
Memory Allocation	Memory is allocated at compile time (static allocation).	Memory is allocated at runtime (dynamic allocation).

Memory Layout	Elements are stored in continuous blocks of memory.	Nodes are scattered in memory and linked through pointers.
Size	Size is fixed once declared. It cannot be changed during execution.	Size is dynamic. Nodes can be added or removed during execution.
Access Method	Random access possible using index (arr[i]). Access time is O(1).	Sequential access only. Need to traverse from head node. Access time is O(n).
Insertion	Insertion at middle requires shifting elements, which is time-consuming.	Insertion is easy. Only pointer changes are required. No shifting needed.
Deletion	Deletion requires shifting elements to maintain order.	Deletion is easy. Only pointer adjustment is required.
Memory Utilization	May cause memory wastage if declared size is larger than required.	Efficient memory usage because memory is allocated as needed.
Extra Memory	No extra memory required except the array elements.	Extra memory required for storing pointer(s) in each node.
Implementation Complexity	Simple and easy to implement.	Slightly complex due to pointer handling.

4. Differentiate Linear and Non-Linear Data Structures.

Ans.:

Basis	Linear Data Structure	Non-Linear Data Structure
Definition	A data structure in which elements are arranged sequentially one after another.	A data structure in which elements are arranged hierarchically or interconnected.
Relationship Between Elements	Each element has a unique predecessor and successor (except first and last).	One element can have multiple predecessors or successors.
Levels	Single level organization.	Multi-level organization.
Traversal	Traversed sequentially in a single run.	Requires special traversal techniques like DFS or BFS.
Implementation	Easier to implement and understand.	More complex implementation.
Memory Structure	Usually sequential memory arrangement.	Memory arranged in hierarchical or network form.
Examples	Array, Stack, Queue, Linked List.	Tree, Graph.
Real-world Usage	Used when data processing is sequential.	Used for hierarchical or network-based relationships.



---

5. Compare arrays and linked lists as examples of static and dynamic data structures.

Ans.:

Basis	Array (Static Data Structure)	Linked List (Dynamic Data Structure)
Nature	Static data structure	Dynamic data structure
Memory Allocation	Memory allocated at compile time or before execution	Memory allocated at runtime using dynamic memory allocation (malloc/free in C)
Memory Location	Contiguous memory locations	Non-contiguous memory locations

Size	Fixed size, cannot be changed after declaration	Size can grow or shrink during execution
Flexibility	Less flexible	Highly flexible
Memory Utilization	May lead to memory wastage if declared size is larger	Efficient memory utilization (allocates as needed)
Insertion	Costly due to shifting of elements	Easy, only pointer adjustment required
Deletion	Costly due to shifting	Easy, pointer modification required
Access Time	Direct access using index (O(1))	Sequential access only (O(n))
Implementation	Simple to implement	Slightly complex due to pointer handling