### a) What is data structure?

A data structure is a specialized format for organizing, managing, and storing data efficiently for operations like access, retrieval, and modification.

- Arrays  
- Linked Lists  
- Stacks  
- Trees

### b) What is sorting? State the techniques of sorting.

Sorting is arranging elements in a specific order (ascending/descending).

- Bubble Sort  
- Insertion Sort  
- Selection Sort  
- Merge Sort  
- Quick Sort  
- Heap Sort

### c) What is a non-primitive data structure?

Non-primitive data structures are complex structures derived from primitive types.

- Arrays  
- Stacks  
- Queues  
- Linked Lists  
- Trees

### d) What is searching?

Searching is the process of finding a specific element in a data structure.

- Linear Search  
- Binary Search  
- Hashing

### e) Mention the features of ADT (Abstract Data Type).

ADT defines data operations independently of implementation.

- Abstraction  
- Modularity  
- Encapsulation  
- User-defined behavior

### f) What are the types of linked lists?

1. Singly Linked List  
2. Doubly Linked List  
3. Circular Linked List  
4. Circular Doubly Linked List

### g) List down the applications of a list.

1. Dynamic memory allocation  
2. Implementation of stacks and queues  
3. Symbol tables in compilers  
4. Adjacency lists in graphs

### h) What is a polynomial? How is it represented?

A polynomial is a mathematical expression with variables and coefficients. It is represented using arrays (coefficients and powers) or linked lists (nodes with coefficients and powers).

### i) Differentiate array & structure.

- \*\*Array:\*\* Collection of elements of the same type, stored contiguously.  
- \*\*Structure:\*\* Collection of variables of different types, grouped together.

### j) What are the applications of stack?

1. Expression evaluation (prefix, postfix)  
2. Backtracking (e.g., mazes, recursion)  
3. Parsing and syntax checking  
4. Function call management

### Q2 a) Explain different types of Dynamic Memory Allocation functions.

Dynamic memory allocation functions are used to manage memory during runtime.

- malloc(): Allocates memory but doesn't initialize.  
- calloc(): Allocates memory and initializes it to zero.  
- realloc(): Adjusts memory size dynamically.  
- free(): Deallocates previously allocated memory.

### Q2 b) Explain Linear Data Structure with examples.

Linear data structures arrange data sequentially. Examples include:

- Arrays: Fixed size, indexed.  
- Linked Lists: Dynamic size, nodes linked via pointers.  
- Stacks and Queues: LIFO and FIFO operations.

### Q2 c) What is stack? Explain different operations used in stack.

A stack is a LIFO (Last In, First Out) data structure. Common operations are:

- Push: Add element to the top.  
- Pop: Remove the top element.  
- Peek: View the top element without removing it.

### Q2 d) What is an algorithm? Explain its characteristics.

An algorithm is a step-by-step method to solve a problem. Characteristics include:

- Input/Output: Defined inputs and outputs.  
- Finiteness: Terminates after a finite number of steps.  
- Correctness: Produces the expected result.

### Q2 e) Explain selection sort technique with example.

Selection sort repeatedly selects the smallest element and places it in sorted order.

- Array: [4, 3, 1, 2]  
Steps: [1, 3, 4, 2] → [1, 2, 4, 3] → [1, 2, 3, 4].

### Q3 a) Write a function to create and display a singly linked list.

A singly linked list contains nodes connected linearly with one pointer per node. Example code:

- void display(Node\* head) {  
 Node\* temp = head;  
 while (temp) {  
 printf("%d -> ", temp->data);  
 temp = temp->next;  
 }  
 printf("NULL");  
}

### Q3 b) Write a function to insert an element into a queue using an array.

Elements can be inserted at the rear of the queue. Example code:

- void enqueue(int queue[], int \*rear, int value) {  
 queue[++(\*rear)] = value;  
}

### Q3 c) Explain BFS traversing technique with an example.

Breadth-First Search (BFS) visits all nodes at the current level before moving to the next.

- Example for graph starting at node 1:  
Graph: {1 → 2, 3 → 4, 5}  
BFS Order: 1 → 2 → 3 → 4 → 5.

### Q3 d) Write a function for preorder traversal of a tree.

Preorder traversal visits the root, then left subtree, and finally right subtree. Example code:

- void preorder(Node\* root) {  
 if (root) {  
 printf("%d ", root->data);  
 preorder(root->left);  
 preorder(root->right);  
 }  
}

### Q3 e) Write an algorithm to convert an infix expression to postfix.

Algorithm:  
1. Scan the expression from left to right.  
2. If operand, add to output.  
3. If operator, push to stack (respect precedence).  
4. Pop operators to output until higher precedence is found.  
5. Add remaining stack elements to output.

### Q3 e) Write an algorithm to convert an infix expression to postfix.

Algorithm:  
1. Scan the expression from left to right.  
2. If operand, add to output.  
3. If operator, push to stack (respect precedence).  
4. Pop operators to output until higher precedence is found.  
5. Add remaining stack elements to output.

### Q4 a) Construct an AVL tree for the given data.

Steps:  
1. Insert nodes sequentially (20, 10, 30...).  
2. Perform rotations to balance the tree.

- Final AVL Tree (Level Order): 20 → 10 → 30 → 5 → 15 → 25 → 35 → 13 → 17.

### Q4 b) Construct a Binary Search Tree (BST).

Steps:  
Insert nodes sequentially into a tree such that left < root < right.

- Final BST (In-Order): 2 → 10 → 13 → 29 → 57 → 61 → 78 → 95.

### Q4 c) Sort the data using selection sort.

Array: [12, 11, 13, 5, 6]  
Steps:  
1. [5, 11, 13, 12, 6]  
2. [5, 6, 13, 12, 11]  
3. [5, 6, 11, 12, 13]

### Q4 d) Write a C program to display a linked list in reverse order.

Example code:

- #include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Node {  
 int data;  
 struct Node\* next;  
} Node;  
  
void reversePrint(Node\* head) {  
 if (head == NULL) return;  
 reversePrint(head->next);  
 printf("%d ", head->data);  
}  
  
int main() {  
 Node\* head = (Node\*)malloc(sizeof(Node));  
 head->data = 10;  
 head->next = NULL; // Add more nodes as needed  
 reversePrint(head);  
 return 0;  
}

### Q4 e) What is a graph? Explain its representation techniques.

A graph is a data structure with nodes (vertices) and edges (connections).

- Adjacency Matrix: 2D array where cell (i, j) indicates edge existence. Space complexity: O(V²).  
- Adjacency List: Array of linked lists for each vertex. Efficient for sparse graphs.  
- Edge List: List of all edges with start and end vertices.

### Q5 a) Convert the following expressions into prefix:

i) A + B \* C \* (D - A) ^ F ^ H  
Steps:  
1. Convert to postfix: A B C \* D A - F H ^ ^ \* +.  
2. Convert postfix to prefix: + A \* B \* C ^ ^ D A - F H.  
  
ii) A \* (B \* C + D \* E) + F  
Steps:  
1. Convert to postfix: A B C \* D E \* + F +.  
2. Convert postfix to prefix: + \* A + \* B C \* D E F.

### Q5 b) Define the following terms:

i) Parent Node: A node in a tree that has child nodes connected to it.  
ii) Subtree: A smaller tree derived from a node and its descendants in the main tree.  
iii) Directed Graph: A graph where edges have a direction, connecting one vertex to another specifically (e.g., (A → B)).

### Q5 c) Indegree and Outdegree of each vertex in the graph:

1. Indegree: Number of incoming edges.  
2. Outdegree: Number of outgoing edges.