1. **Write a program in C to Check if a particular customer account ID exists in the list using Linear Search.**

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int accountIDs[50];
   int n, i, searchID, found = 0;
   clrscr(); // clear the screen (Turbo C specific)
   printf("Enter the number of customer accounts: ");
   scanf("%d", &n);

   printf("\nEnter %d Customer Account IDs:\n", n);
   for (i = 0; i < n; i++)
   {
      printf("Account ID [%d]: ", i + 1);
      scanf("%d", &accountIDs[i]);
   }
   printf("\nEnter the Account ID to search: ");
   scanf("%d", &searchID);
   // Linear Search
   for (i = 0; i < n; i++)
   {
      if (accountIDs[i] == searchID)
      {
         found = 1;
         break;
      }
   }
   if (found == 1)
      printf("\nAccount ID %d FOUND at position %d.\n", searchID, i +
1);
   else
      printf("\nAccount ID %d NOT FOUND in the list.\n", searchID);

   getch(); // wait for keypress before exit (Turbo C specific)
}
```

2. **Write a program in C to Sort the salaries using the bubble sort algorithm.**

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int n, i, j;
   float salary[50], temp;
   clrscr(); // clear screen (Turbo C specific)
   printf("Enter the number of employees: ");
   scanf("%d", &n);
   printf("\nEnter the salaries of %d employees:\n", n);
   for (i = 0; i < n; i++)
   {
      printf("Salary [%d]: ", i + 1);
      scanf("%f", &salary[i]);
   }

   // Bubble Sort Algorithm
   for (i = 0; i < n - 1; i++)
   {
      for (j = 0; j < n - i - 1; j++)
      {
         if (salary[j] > salary[j + 1])
         {
            temp = salary[j];
            salary[j] = salary[j + 1];
            salary[j + 1] = temp;
         }
      }
   }
   printf("\nSalaries in ascending order:\n");
   for (i = 0; i < n; i++)
   {
      printf("%.2f\n", salary[i]);
   }
   getch(); // wait for key press before exit (Turbo C specific)
}
```

3. **Write a program in C to Implement push, pop, peek operations using array-based stack data structure.**

```
#include <stdio.h>
#include <conio.h>
#define MAX 50   // Maximum size of the stack

int stack[MAX];
int top = -1;

// Function prototypes
void push();
void pop();
void peek();
void display();

void main()
{
   int choice;
   clrscr(); // Clear screen (Turbo C specific)

   while (1)
   {
      printf("\n--- Stack Operations using Array ---\n");
      printf("1. PUSH\n");
      printf("2. POP\n");
      printf("3. PEEK (View Top Element) \n");
      printf("4. DISPLAY Stack\n");
      printf("5. EXIT\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
         push();
         break;
      case 2:
         pop();
         break;
      case 3:
         peek();
         break;
```

```
        case 4:
            display();
            break;
        case 5:
            printf("\nExiting program...\n");
            getch();
            return;
        default:
            printf("\nInvalid choice! Please try again.\n");
        }
    }
}

// Function to push an element into the stack
void push()
{
    int value;
    if (top == MAX - 1)
    {
        printf("\nStack Overflow! Cannot push more elements.\n");
    }
    else
    {
        printf("Enter value to PUSH: ");
        scanf("%d", &value);
        top++;
        stack[top] = value;
        printf("Value %d pushed onto the stack.\n", value);
    }
}

// Function to pop the top element from the stack
void pop()
{
    if (top == -1)
    {
        printf("\nStack Underflow! No elements to POP.\n");
    }
    else
    {
        printf("Value %d popped from the stack.\n", stack[top]);
        top--;
    }
```

```
}

// Function to peek (view) the top element
void peek()
{
   if (top == -1)
   {
      printf("\nStack is empty! No top element.\n");
   }
   else
   {
      printf("Top element is: %d\n", stack[top]);
   }
}


// Function to display all stack elements
void display()
{
   int i;
   if (top == -1)
   {
      printf("\nStack is empty!\n");
   }
   else
   {
      printf("\nStack elements are:\n");
      for (i = top; i >= 0; i--)
      {
         printf("%d\n", stack[i]);
      }
   }
}
```

4. **Write a program in C to Implement enqueue and dequeue operations using queue data structure.**

```
#include <stdio.h>
#include <conio.h>
#define MAX 50   // Maximum size of the queue

int queue[MAX];
int front = -1, rear = -1;

// Function prototypes
void enqueue();
void dequeue();
void display();

void main()
{
   int choice;
   clrscr(); // Clear screen (Turbo C specific)

   while (1)
   {
      printf("\n--- Queue Operations using Array ---\n");
      printf("1. ENQUEUE (Insert)\n");
      printf("2. DEQUEUE (Delete)\n");
      printf("3. DISPLAY Queue\n");
      printf("4. EXIT\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
         enqueue();
         break;
      case 2:
         dequeue();
         break;
      case 3:
         display();
         break;
      case 4:
         printf("\nExiting program...\n");
```

```
            getch();
            return;
        default:
            printf("\nInvalid choice! Please try again.\n");
        }
    }
}

// Function to insert an element in the queue
void enqueue()
{
    int value;
    if (rear == MAX - 1)
    {
        printf("\nQueue Overflow! Cannot enqueue more elements.\n");
    }
    else
    {
        printf("Enter value to ENQUEUE: ");
        scanf("%d", &value);

        if (front == -1)
            front = 0;

        rear++;
        queue[rear] = value;
        printf("Value %d enqueued into the queue.\n", value);
    }
}

// Function to delete an element from the queue
void dequeue()
{
    if (front == -1 || front > rear)
    {
        printf("\nQueue Underflow! No elements to dequeue.\n");
    }
    else
    {
        printf("Value %d dequeued from the queue.\n", queue[front]);
        front++;

        if (front > rear) // Reset queue when it becomes empty
```

```
        {
            front = rear = -1;
        }
    }
}

// Function to display the queue
void display()
{
    int i;
    if (front == -1)
    {
        printf("\nQueue is empty!\n");
    }
    else
    {
        printf("\nQueue elements are:\n");
        for (i = front; i <= rear; i++)
        {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
```

5. Write a program in C to Create, traverse nodes in a singly linked list.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>  // Turbo C uses alloc.h instead of stdlib.h for
malloc()

// Structure for a singly linked list node
struct Node
{
   int data;
   struct Node *next;
};

void main()
{
   struct Node *head = NULL, *newNode, *temp;
   int choice = 1;
   int value;

   clrscr(); // Clear screen (Turbo C specific)
   printf("=== Singly Linked List Creation and Traversal ===\n");

   // Linked List Creation
   while (choice)
   {
      newNode = (struct Node *)malloc(sizeof(struct Node));
      if (newNode == NULL)
      {
         printf("Memory allocation failed!\n");
         break;
      }

      printf("\nEnter data for the node: ");
      scanf("%d", &value);
      newNode->data = value;
      newNode->next = NULL;

      if (head == NULL)
      {
         head = newNode;
         temp = head;
      }
```

```
        else
        {
            temp->next = newNode;
            temp = newNode;
        }

        printf("Do you want to add another node? (1 = Yes / 0 = No): ");
        scanf("%d", &choice);
    }

    // Traversing the linked list
    printf("\nLinked List Elements:\n");
    temp = head;
    if (temp == NULL)
    {
        printf("List is empty!\n");
    }
    else
    {
        while (temp != NULL)
        {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }

    getch(); // Wait for key press before exiting
}
```

6. **Write a program in C to Create a adjacency list using graph data structure.**

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>  // for malloc() in Turbo C

// Structure for adjacency list node
struct Node
{
   int vertex;
   struct Node *next;
};

// Function to create a new node
struct Node* createNode(int v)
{
   struct Node *newNode;
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}

void main()
{
   struct Node *adjList[20]; // Array of pointers for adjacency list
   struct Node *temp;
   int vertices, edges;
   int i, j, src, dest;

   clrscr(); // Clear screen

   printf("=== Create Graph using Adjacency List ===\n");
   printf("Enter number of vertices: ");
   scanf("%d", &vertices);

   // Initialize adjacency list to NULL
   for (i = 0; i < vertices; i++)
   {
      adjList[i] = NULL;
   }
```

```
    printf("Enter number of edges: ");
    scanf("%d", &edges);

    printf("\nEnter edges (source destination):\n");
    for (i = 0; i < edges; i++)
    {
        printf("Edge %d: ", i + 1);
        scanf("%d %d", &src, &dest);

        // Create node for destination vertex
        temp = createNode(dest);
        temp->next = adjList[src];
        adjList[src] = temp;

        // (Optional) For undirected graph, add reverse edge
        // Uncomment this part if you want an undirected graph:
        /*
        temp = createNode(src);
        temp->next = adjList[dest];
        adjList[dest] = temp;
        */
    }

    // Display the adjacency list
    printf("\nAdjacency List Representation:\n");
    for (i = 0; i < vertices; i++)
    {
        struct Node *ptr = adjList[i];
        printf("Vertex %d: ", i);
        while (ptr != NULL)
        {
            printf("-> %d ", ptr->vertex);
            ptr = ptr->next;
        }
        printf("-> NULL\n");
    }

    getch(); // Wait for keypress before exit
}
```

7. **Write a program in C to Create a adjacency matrix using graph data structure.**

```
#include <stdio.h>
#include <conio.h>

void main()
{
   int adj[20][20];   // adjacency matrix (max 20 vertices)
   int vertices, edges;
   int i, j, src, dest;

   clrscr(); // clear screen

   printf("=== Create Graph using Adjacency Matrix ===\n");
   printf("Enter number of vertices: ");
   scanf("%d", &vertices);

   printf("Enter number of edges: ");
   scanf("%d", &edges);

   // Initialize all entries to 0
   for (i = 0; i < vertices; i++)
   {
      for (j = 0; j < vertices; j++)
      {
         adj[i][j] = 0;
      }
   }

   printf("\nEnter edges (source destination):\n");
   for (i = 0; i < edges; i++)
   {
      printf("Edge %d: ", i + 1);
      scanf("%d %d", &src, &dest);

      adj[src][dest] = 1;  // For directed graph

      // Uncomment the next line for undirected graph:
      // adj[dest][src] = 1;
   }

   // Display adjacency matrix
   printf("\nAdjacency Matrix Representation:\n\n");
```

```
    printf("    ");
    for (i = 0; i < vertices; i++)
        printf("%3d", i);
    printf("\n");

    for (i = 0; i < vertices; i++)
    {
        printf("%3d ", i);
        for (j = 0; j < vertices; j++)
        {
            printf("%3d", adj[i][j]);
        }
        printf("\n");
    }

    getch(); // Wait for key press before exit
}
```

8. Write a program in C to Create a graph of al-least 5 nodes find and print BFS.

```
#include <stdio.h>
#include <conio.h>
#define MAX 20

int adj[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = -1, rear = -1;

void enqueue(int v)
{
   if (rear == MAX - 1)
      printf("Queue Overflow!\n");
   else
   {
      if (front == -1)
         front = 0;
      rear++;
      queue[rear] = v;
   }
}

int dequeue()
{
   int v;
   if (front == -1 || front > rear)
      return -1;
   v = queue[front];
   front++;
   return v;
}

void BFS(int start, int vertices)
{
   int i, current;

   for (i = 0; i < vertices; i++)
      visited[i] = 0;

   enqueue(start);
```

```
    visited[start] = 1;

    printf("\nBFS Traversal starting from node %d: ", start);

    while ((current = dequeue()) != -1)
    {
        printf("%d ", current);

        for (i = 0; i < vertices; i++)
        {
            if (adj[current][i] == 1 && visited[i] == 0)
            {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}
void main()
{
    int vertices, edges;
    int i, j, src, dest, start;
    clrscr(); // clear screen
    printf("=== Create Graph and Perform BFS Traversal ===\n");
    printf("Enter number of vertices (at least 5): ");
    scanf("%d", &vertices);
    if (vertices < 5)
    {
        printf("Graph must have at least 5 nodes!\n");
        getch();
        return;
    }

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Initialize adjacency matrix
    for (i = 0; i < vertices; i++)
        for (j = 0; j < vertices; j++)
            adj[i][j] = 0;

    printf("\nEnter edges (source destination):\n");
    for (i = 0; i < edges; i++)
```

```
    {
        printf("Edge %d: ", i + 1);
        scanf("%d %d", &src, &dest);
        adj[src][dest] = 1;

        // Uncomment this for undirected graph:
        // adj[dest][src] = 1;
    }

    printf("\nEnter starting node for BFS: ");
    scanf("%d", &start);

    BFS(start, vertices);
    getch();
}
```

9. Write a program in C to Construct an expression tree from the given prefix expression, e.g., +--a*bc/def, traverse it using post-order traversal.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <string.h>

// Structure for tree node
struct Node
{
   char data;
   struct Node *left, *right;
};

// Function to create a new tree node
struct Node *createNode(char data)
{
   struct Node *newNode;
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

// Check if character is an operator
int isOperator(char ch)
{
   if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
      return 1;
   return 0;
}

// Stack implementation for tree nodes
struct Node *stack[50];
int top = -1;

void push(struct Node *node)
{
   stack[++top] = node;
}
```

```
struct Node *pop()
{
   if (top == -1)
      return NULL;
   return stack[top--];
}


// Function to construct expression tree from prefix expression
struct Node *constructTree(char prefix[])
{
   int i;
   int len = strlen(prefix);
   struct Node *node, *node1, *node2;

   // Traverse prefix expression from right to left
   for (i = len - 1; i >= 0; i--)
   {
      // If operand, create node and push
      if (!isOperator(prefix[i]))
      {
         node = createNode(prefix[i]);
         push(node);
      }
      else
      {
         // Operator: pop two nodes and make them children
         node = createNode(prefix[i]);
         node1 = pop();
         node2 = pop();

         node->left = node1;
         node->right = node2;

         push(node);
      }
   }

   // Remaining node in stack is the root
   return pop();
}
```

```
// Postorder traversal (Left → Right → Root)
void postorder(struct Node *root)
{
   if (root == NULL)
      return;
   postorder(root->left);
   postorder(root->right);
   printf("%c", root->data);
}

void main()
{
   char prefix[50];
   struct Node *root;

   clrscr(); // Clear screen (Turbo C specific)

   printf("=== Expression Tree Construction (Prefix to Postorder)
===\n");
   printf("Enter a Prefix Expression (e.g., +--a*bc/def): ");
   scanf("%s", prefix);

   root = constructTree(prefix);

   printf("\nPostorder Traversal of Expression Tree: ");
   postorder(root);

   getch();
}
```

10. Write a program in C to Implement a hash table of size 10 and use the division method as a hash function. 1. Search (key): Search for the value associated with a given key.

```
#include <stdio.h>
#include <conio.h>

#define SIZE 10  // Size of hash table

// Structure for each element in the hash table
struct HashItem
{
   int key;
   int value;
};

// Declare hash table
struct HashItem hashTable[SIZE];

// Function to initialize hash table
void initHashTable()
{
   int i;
   for (i = 0; i < SIZE; i++)
   {
      hashTable[i].key = -1;   // -1 indicates empty slot
      hashTable[i].value = -1;
   }
}

// Hash function using division method
int hashFunction(int key)
{
   return key % SIZE;
}

// Insert key-value pair
void insert(int key, int value)
{
   int index = hashFunction(key);
   int startIndex = index;

   while (hashTable[index].key != -1)
```

```
    {
        index = (index + 1) % SIZE;
        if (index == startIndex)
        {
            printf("\nHash Table is full! Cannot insert more.\n");
            return;
        }
    }

    hashTable[index].key = key;
    hashTable[index].value = value;

    printf("\nInserted (Key: %d, Value: %d) at Index %d\n", key, value,
index);
}

// Search for a key
void search(int key)
{
    int index = hashFunction(key);
    int startIndex = index;

    while (hashTable[index].key != -1)
    {
        if (hashTable[index].key == key)
        {
            printf("\nKey %d found at Index %d with Value = %d\n", key,
index, hashTable[index].value);
            return;
        }

        index = (index + 1) % SIZE;
        if (index == startIndex)
            break;
    }

    printf("\nKey %d not found in the hash table.\n", key);
}

// Display hash table
void display()
{
    int i;
```

```
    printf("\n--- Hash Table ---\n");
    printf("Index\tKey\tValue\n");
    for (i = 0; i < SIZE; i++)
    {
        if (hashTable[i].key != -1)
            printf("%d\t%d\t%d\n", i, hashTable[i].key, hashTable[i].value);
        else
            printf("%d\t--\t--\n", i);
    }
}

void main()
{
    int choice, key, value;

    clrscr(); // Clear screen

    initHashTable();

    while (1)
    {
        printf("\n=== Hash Table using Division Method ===\n");
        printf("1. Insert (key, value)\n");
        printf("2. Search (key)\n");
        printf("3. Display Table\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("\nEnter Key (integer): ");
            scanf("%d", &key);
            printf("Enter Value (integer): ");
            scanf("%d", &value);
            insert(key, value);
            break;

        case 2:
            printf("\nEnter Key to Search: ");
            scanf("%d", &key);
            search(key);
```

```
            break;

        case 3:
            display();
            break;

        case 4:
            printf("\nExiting program...\n");
            getch();
            return;

        default:
            printf("\nInvalid choice! Please try again.\n");
        }
    }
}
```