A 1. Create an array to store student names in a class, allowing insertion, deletion, and traversal operations.

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

char names[10][20];
int count = 0;

void insert() {
    if(count == 10) {
        printf("\nList is Full");
    } else {
        printf("\nEnter Name: ");
        scanf("%s", names[count]);
        count++;
        printf("\nInserted Successfully");
    }
}

void deleteName() {
    int i, pos;
    if(count == 0) {
        printf("\nList is Empty");
    } else {
        printf("\nEnter Position to Delete: ");
        scanf("%d", &pos);

        if(pos > count || pos < 1) {
            printf("\nInvalid Position");
        } else {
            for(i = pos - 1; i < count - 1; i++) {
                strcpy(names[i], names[i + 1]);
            }
            count--;
            printf("\nDeleted Successfully");
        }
    }
}

void display() {
    int i;
    if(count == 0) {
        printf("\nList is Empty");
    } else {
        printf("\nStudent List:\n");
        for(i = 0; i < count; i++) {
            printf("%d. %s\n", i + 1, names[i]);
        }
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Add Student");
        printf("\n2. Delete Student");
        printf("\n3. Display List");
        printf("\n4. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: insert(); break;
            case 2: deleteName(); break;
            case 3: display(); break;
            case 4: return;
            default: printf("\nInvalid Choice");
        }
    }
}
```

A 2. Represent a railway timetable as a sparse matrix where only active train schedules are stored efficiently.

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

struct Schedule {
    int train_id;
    int station_id;
    int time;
};

void main() {
    struct Schedule table[20];
    int i, total_trains, total_stations, count;

    clrscr();

    printf("Enter Total Trains: ");
    scanf("%d", &total_trains);
    printf("Enter Total Stations: ");
    scanf("%d", &total_stations);

    printf("Enter Number of Active Stops: ");
    scanf("%d", &count);

    table[0].train_id = total_trains;
    table[0].station_id = total_stations;
    table[0].time = count;

    printf("\nEnter Details (TrainID StationID Time):\n");
    for(i = 1; i <= count; i++) {
        printf("Stop %d: ", i);
        scanf("%d %d %d", &table[i].train_id, &table[i].station_id, &table[i].time);
    }

    printf("\n\n--- Railway Sparse Matrix ---\n");
    printf("Train\tStation\tTime\n");
    printf("------------------------\n");

    printf("%d\t%d\t%d\n", table[0].train_id, table[0].station_id, table[0].time);

    for(i = 1; i <= count; i++) {
        printf("%d\t%d\t%d\n", table[i].train_id, table[i].station_id, table[i].time);
    }

    getch();
}
```

B 1. Implement a student record system using a singly linked list that supports creating, inserting, deleting, and displaying student records.

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

struct Student {
    int roll;
    char name[20];
    struct Student *next;
} *head = NULL;

void insert() {
    struct Student *newNode, *temp;
    newNode = (struct Student*)malloc(sizeof(struct Student));

    printf("\nEnter Roll No: ");
    scanf("%d", &newNode->roll);
    printf("Enter Name: ");
    scanf("%s", newNode->name);
    newNode->next = NULL;

    if(head == NULL) {
        head = newNode;
    } else {
        temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("\nStudent Added Successfully");
}

void deleteStudent() {
    struct Student *temp, *prev;
    int roll;

    if(head == NULL) {
        printf("\nList is Empty");
        return;
    }

    printf("\nEnter Roll No to Delete: ");
    scanf("%d", &roll);

    temp = head;

    if(temp->roll == roll) {
        head = temp->next;
        free(temp);
        printf("\nDeleted Successfully");
        return;
    }

    while(temp != NULL && temp->roll != roll) {
        prev = temp;
        temp = temp->next;
    }

    if(temp == NULL) {
        printf("\nStudent Not Found");
    } else {
        prev->next = temp->next;
        free(temp);
        printf("\nDeleted Successfully");
    }
}

void display() {
    struct Student *temp;
    temp = head;

    if(temp == NULL) {
        printf("\nList is Empty");
    } else {
        printf("\n--- Student Records ---\n");
        while(temp != NULL) {
            printf("Roll: %d | Name: %s\n", temp->roll, temp->name);
            temp = temp->next;
        }
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Add Student");
        printf("\n2. Delete Student");
        printf("\n3. Display Records");
        printf("\n4. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: insert(); break;
            case 2: deleteStudent(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nInvalid Choice");
        }
    }
}
```

B 2. Write a program to store a sequence of train stations in a singly linked list and reverse the order to simulate a return journey.

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

struct Station {
    char name[30];
    struct Station *next;
} *head = NULL;

void addStation() {
    struct Station *newNode, *temp;

    newNode = (struct Station*)malloc(sizeof(struct Station));
    printf("\nEnter Station Name: ");
    scanf("%s", newNode->name);
    newNode->next = NULL;

    if(head == NULL) {
        head = newNode;
    } else {
        temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("\nStation Added");
}

void reverseRoute() {
    struct Station *prev = NULL;
    struct Station *current = head;
    struct Station *next = NULL;

    if(head == NULL) {
        printf("\nList is Empty");
        return;
    }

    while(current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    printf("\nRoute Reversed (Return Journey Ready)");
}

void display() {
    struct Station *temp = head;
    if(temp == NULL) {
        printf("\nNo Stations Added");
    } else {
        printf("\n--- Train Route ---\n");
        while(temp != NULL) {
            printf("%s -> ", temp->name);
            temp = temp->next;
        }
        printf("END\n");
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Add Station");
        printf("\n2. Reverse Route (Return Journey)");
        printf("\n3. View Route");
        printf("\n4. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: addStation(); break;
            case 2: reverseRoute(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nInvalid Choice");
        }
    }
}
```

B 4. Implement a music playlist where songs can be played forward and backward using a doubly linked list.

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

```
struct Song {
    char title[20];
    struct Song *next;
    struct Song *prev;
} *head = NULL;

void addSong() {
    struct Song *newNode, *temp;

    newNode = (struct Song*)malloc(sizeof(struct Song));
    printf("\nEnter Song Title: ");
    scanf("%s", newNode->title);
    newNode->next = NULL;
    newNode->prev = NULL;

    if(head == NULL) {
        head = newNode;
    } else {
        temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
    printf("\nSong Added to Playlist");
}

void playForward() {
    struct Song *temp = head;

    if(head == NULL) {
        printf("\nPlaylist is Empty");
    } else {
        printf("\n--- Playing Forward (First to Last) ---\n");
        while(temp != NULL) {
            printf("Playing: %s\n", temp->title);
            temp = temp->next;
        }
    }
}

void playBackward() {
    struct Song *temp = head;

    if(head == NULL) {
        printf("\nPlaylist is Empty");
        return;
    }
    /* Go to the last song first */
    while(temp->next != NULL) {
        temp = temp->next;
    }

    printf("\n--- Playing Backward (Last to First) ---\n");
    while(temp != NULL) {
        printf("Playing: %s\n", temp->title);
        temp = temp->prev;
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Add Song");
        printf("\n2. Play Forward");
        printf("\n3. Play Backward");
        printf("\n4. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: addSong(); break;
            case 2: playForward(); break;
            case 3: playBackward(); break;
            case 4: exit(0);
            default: printf("\nInvalid Choice");
        }
    }
}
```

C 1. Implement a stack-based text editor feature that supports undo/redo operations.

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

char undoStack[50];
char redoStack[50];
int u_top = -1;
int r_top = -1;
```

```
void type() {
    char c;
    if(u_top == 49) {
        printf("\nBuffer Full");
    } else {
        printf("\nEnter Character to Type: ");
        /* Space before %c skips whitespace */
        scanf(" %c", &c);

        u_top++;
        undoStack[u_top] = c;

        /* Reset Redo Stack on new input */
        r_top = -1;
        printf("\nTyped '%c'", c);
    }
}

void undo() {
    char c;
    if(u_top == -1) {
        printf("\nNothing to Undo");
    } else {
        c = undoStack[u_top];
        u_top--;

        r_top++;
        redoStack[r_top] = c;
        printf("\nUndid '%c'", c);
    }
}

void redo() {
    char c;
    if(r_top == -1) {
        printf("\nNothing to Redo");
    } else {
        c = redoStack[r_top];
        r_top--;

        u_top++;
        undoStack[u_top] = c;
        printf("\nRedid '%c'", c);
    }
}

void display() {
    int i;
    if(u_top == -1) {
        printf("\n[Empty Document]");
    } else {
        printf("\n--- Current Text ---\n");
        for(i = 0; i <= u_top; i++) {
            printf("%c", undoStack[i]);
        }
        printf("\n--------------------");
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Type Character");
        printf("\n2. Undo");
        printf("\n3. Redo");
        printf("\n4. View Text");
        printf("\n5. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: type(); break;
            case 2: undo(); break;
            case 3: redo(); break;
            case 4: display(); break;
            case 5: exit(0);
            default: printf("\nInvalid Choice");
        }
    }
}
```

C 2. Use the stack ADT to check whether parentheses in an arithmetic expression are balanced.

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

char stack[50];
int top = -1;
```

```
void push(char c) {
    top++;
    stack[top] = c;
}

char pop() {
    char c;
    if(top == -1) {
        return '\0';
    }
    c = stack[top];
    top--;
    return c;
}

int match(char a, char b) {
    if(a == '(' && b == ')') return 1;
    if(a == '{' && b == '}') return 1;
    if(a == '[' && b == ']') return 1;
    return 0;
}

void checkBalance() {
    char expr[50], temp;
    int i, valid = 1;

    /* Reset stack for new check */
    top = -1;

    printf("\nEnter Expression: ");
    scanf("%s", expr);

    for(i = 0; i < strlen(expr); i++) {
        if(expr[i] == '(' || expr[i] == '{' || expr[i] == '[') {
            push(expr[i]);
        }
        else if(expr[i] == ')' || expr[i] == '}' || expr[i] == ']') {
            if(top == -1) {
                valid = 0;
                break;
            }
            temp = pop();
            if(!match(temp, expr[i])) {
                valid = 0;
                break;
            }
        }
    }
    if(top != -1) {
        valid = 0;
    }

    if(valid == 1) {
        printf("\nBalanced Expression");
    } else {
        printf("\nUnbalanced Expression");
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Check Expression");
        printf("\n2. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: checkBalance(); break;
            case 2: exit(0);
            default: printf("\nInvalid Choice");
        }
    }
}
```

C 3. Implement a queue to simulate a customer service system where customers join and leave in FIFO order.

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

#define SIZE 5

int queue[SIZE];
int front = -1;
int rear = -1;

void joinQueue() {
    int id;
    if(rear == SIZE - 1) {
        printf("\nQueue is Full (Wait)");
    } else {
        if(front == -1) {
            front = 0;
        }
        printf("\nEnter Customer ID: ");
        scanf("%d", &id);
        rear++;
        queue[rear] = id;
        printf("\nCustomer %d Joined", id);
    }
}

void serveCustomer() {
    if(front == -1 || front > rear) {
        printf("\nNo Customers to Serve");
    } else {
        printf("\nServing Customer ID: %d", queue[front]);
        front++;

        /* Reset queue if empty to save space */
        if(front > rear) {
            front = -1;
            rear = -1;
        }
    }
}

void display() {
    int i;
    if(front == -1 || front > rear) {
        printf("\nQueue is Empty");
    } else {
        printf("\n--- Waiting Line ---\n");
        for(i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    }
}

void main() {
    int choice;
    clrscr();

    while(1) {
        printf("\n\n1. Customer Join (Enqueue)");
        printf("\n2. Serve Customer (Dequeue)");
        printf("\n3. View Line");
        printf("\n4. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: joinQueue(); break;
            case 2: serveCustomer(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nInvalid Choice");
        }
    }
}
```

D 1. Develop a BST to store and search for student records based on roll numbers.

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

struct Node {
    int roll;
    char name[20];
    struct Node *left;
    struct Node *right;
};

struct Node *createNode(int roll, char *name) {
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->roll = roll;
    strcpy(newNode->name, name);
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node *insert(struct Node *root, int roll, char *name) {
    if(root == NULL) {
        return createNode(roll, name);
    }
    if(roll < root->roll) {
        root->left = insert(root->left, roll, name);
    } else if(roll > root->roll) {
        root->right = insert(root->right, roll, name);
    } else {
        printf("\nDuplicate Roll Number Not Allowed");
    }
    return root;
}

void search(struct Node *root, int roll) {
    if(root == NULL) {
        printf("\nStudent Not Found");
        return;
    }
    if(root->roll == roll) {
        printf("\nFound: Roll %d | Name: %s", root->roll, root->name);
    } else if(roll < root->roll) {
        search(root->left, roll);
    } else {
        search(root->right, roll);
    }
}

void inorder(struct Node *root) {
    if(root != NULL) {
        inorder(root->left);
        printf("Roll: %d | Name: %s\n", root->roll, root->name);
        inorder(root->right);
    }
}

void main() {
    struct Node *root = NULL;
    int choice, roll;
    char name[20];

    clrscr();

    while(1) {
        printf("\n\n1. Add Student");
        printf("\n2. Search Student");
        printf("\n3. Display All (Sorted)");
        printf("\n4. Exit");
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("\nEnter Roll: ");
                scanf("%d", &roll);
                printf("Enter Name: ");
                scanf("%s", name);
                root = insert(root, roll, name);
                break;
        case 2:
          printf("\nEnter Roll to Search: ");
          scanf("%d", &roll);
          search(root, roll);
          break;
        case 3:
          printf("\n--- Student Records ---\n");
          inorder(root);
          break;
        case 4:
          exit(0);
        default:
          printf("\nInvalid Choice");
      }
    }
}
```

D 4. Implement DFS to find paths in a city’s metro rail system using an adjacency list.

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

struct Node {
    int station;
    struct Node *next;
};

struct Node *adj[20];
int visited[20];

void addEdge(int src, int dest) {
    struct Node *newNode;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->station = dest;
    newNode->next = adj[src];
    adj[src] = newNode;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->station = src;
    newNode->next = adj[dest];
    adj[dest] = newNode;
}
```

```
void DFS(int v) {
    struct Node *temp = adj[v];

    visited[v] = 1;
    printf("%d ", v);

    while(temp != NULL) {
        int connectedStation = temp->station;
        if(visited[connectedStation] == 0) {
            DFS(connectedStation);
        }
        temp = temp->next;
    }
}

void main() {
    int i, stations, edges, src, dest, start;
    clrscr();

    for(i = 0; i < 20; i++) {
        adj[i] = NULL;
        visited[i] = 0;
    }

    printf("Enter Number of Stations: ");
    scanf("%d", &stations);

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

    printf("\nEnter Connections (Station1 Station2):\n");
    for(i = 0; i < edges; i++) {
        scanf("%d %d", &src, &dest);
        addEdge(src, dest);
    }

    printf("\nEnter Start Station: ");
    scanf("%d", &start);

    printf("\n--- Metro Path (DFS) ---\n");
    DFS(start);

    getch();
}
```

E 1. A school maintains a list of student names who have registered for an event. Given a student’s name, implement sequential search to check if they have registered.

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

void main() {
    char names[10][20];
    char searchName[20];
    int i, n, found = 0;

    clrscr();

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

    printf("Enter student names:\n");
    for(i = 0; i < n; i++) {
        scanf("%s", names[i]);
    }

    printf("\nEnter name to search: ");
    scanf("%s", searchName);

    for(i = 0; i < n; i++) {
        if(strcmp(names[i], searchName) == 0) {
            found = 1;
            break;
        }
    }

    if(found == 1) {
        printf("\nStudent is Registered.");
    } else {
        printf("\nStudent Not Found.");
    }

    getch();
}
```

E 2. A university stores student roll numbers in a sorted list. Implement binary search to find whether a given roll number exists in the list

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

```
void main() {
    int arr[20];
    int n, i, search, first, last, middle;

    clrscr();

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

    printf("Enter %d Roll Numbers (Sorted Order):\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("\nEnter Roll Number to Search: ");
    scanf("%d", &search);

    first = 0;
    last = n - 1;
    middle = (first + last) / 2;

    while(first <= last) {
        if(arr[middle] < search) {
            first = middle + 1;
        } else if(arr[middle] == search) {
            printf("\nFound at Position %d", middle + 1);
            break;
        } else {
            last = middle - 1;
        }
        middle = (first + last) / 2;
    }

    if(first > last) {
        printf("\nNot Found! %d is not present.", search);
    }

    getch();
}
```

E 5. Implement the Bubble Sort algorithm to sort an array of student marks.

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

void main() {
    int marks[20];
    int n, i, j, temp;

    clrscr();

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

    printf("Enter Marks:\n");
    for(i = 0; i < n; i++) {
        scanf("%d", &marks[i]);
    }

    for(i = 0; i < n - 1; i++) {
        for(j = 0; j < n - i - 1; j++) {
            if(marks[j] > marks[j + 1]) {
                temp = marks[j];
                marks[j] = marks[j + 1];
                marks[j + 1] = temp;
            }
        }
    }

    printf("\nSorted Marks:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", marks[i]);
    }

    getch();
}
```

E 7. Implement the Insertion Sort algorithm to sort student names in ascending order

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

void main() {
    char names[20][20];
    char key[20];
    int n, i, j;

    clrscr();

    printf("Enter Number of Students: ");
    scanf("%d", &n);
    printf("Enter Names:\n");
    for(i = 0; i < n; i++) {
        scanf("%s", names[i]);
    }

    for(i = 1; i < n; i++) {
        strcpy(key, names[i]);
        j = i - 1;

        while(j >= 0 && strcmp(names[j], key) > 0) {
            strcpy(names[j + 1], names[j]);
            j = j - 1;
        }
        strcpy(names[j + 1], key);
    }

    printf("\nSorted Names:\n");
    for(i = 0; i < n; i++) {
        printf("%s\n", names[i]);
    }

    getch();
}
```