

Tab 1

1. To write a menu-based program that stores roll numbers of students who attended a training program and allows searching using Linear and Binary Search algorithms.

```
#include <stdio.h>

// ===== LINEAR SEARCH =====
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key)
            return i;
    }
    return -1;
}

// ===== BINARY SEARCH =====
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key)
            return mid;
        else if (key < arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}

// ===== MAIN =====
int main() {
    int choice, n, key, result;
    int arr[100];

    do {
        printf("\n--- Search Program (Linear / Binary) ---\n");
        printf("1. Linear Search (Unsorted Data)\n");
        printf("2. Binary Search (Sorted Data)\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter number of students: ");
                scanf("%d", &n);
                printf("Enter roll numbers (unsorted):\n");
                for (int i = 0; i < n; i++)
                    scanf("%d", &arr[i]);

                printf("Enter roll number to search: ");
                scanf("%d", &key);
                result = linearSearch(arr, n, key);
                if (result == -1)
                    printf("Roll number %d not found.\n", key);
                else
                    printf("Roll number %d found at index %d.\n", key, result);
        }
    } while (choice != 0);
}
```

```

result = linearSearch(arr, n, key);
if (result != -1)
    printf("Student attended the training. Found at index %d.\n", result);
else
    printf("Student did not attend the training.\n");
break;

case 2:
printf("Enter number of students: ");
scanf("%d", &n);
printf("Enter roll numbers (sorted):\n");
for (int i = 0; i < n; i++)
    scanf("%d", &arr[i]);

printf("Enter roll number to search: ");
scanf("%d", &key);

result = binarySearch(arr, n, key);
if (result != -1)
    printf("Student attended the training. Found at index %d.\n", result);
else
    printf("Student did not attend the training.\n");
break;

case 0:
printf("Exiting...\n");
break;

default:
printf("Invalid choice!\n");
}

} while (choice != 0);

return 0;
}

```

```
===== Training Program Attendance Search =====
```

Menu:

1. Linear Search (Unsorted Roll Numbers)
2. Binary Search (Sorted Roll Numbers)
3. Exit

Enter your choice (1-3): 2

Enter number of students: 4

Enter roll numbers (sorted):

1 2 3 5

Enter roll number to search: 3

Student attended the training. Found at index 2.

Menu:

1. Linear Search (Unsorted Roll Numbers)
2. Binary Search (Sorted Roll Numbers)
3. Exit

Enter your choice (1-3): |

Tab 2

2.To write a C program to store the first-year percentage of students in an array and sort the array in ascending order using Selection Sort and Bubble Sort. The program should also display the top five scores after sorting.

```
#include <stdio.h>

// ===== UTILITY FUNCTIONS =====
void swap(float *a, float *b) {
    float temp = *a;
    *a = *b;
    *b = temp;
}

// ===== SELECTION SORT =====
void selectionSort(float arr[], int n) {
    int minIndex;
    for (int i = 0; i < n - 1; i++) {
        minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        swap(&arr[i], &arr[minIndex]);
    }
}

// ===== BUBBLE SORT =====
void bubbleSort(float arr[], int n) {
    int swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = 0;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
                swapped = 1;
            }
        }
        if (!swapped)
            break;
    }
}

// ===== DISPLAY FUNCTIONS =====
void display(float arr[], int n) {
    printf("Sorted Percentages:\n");
    for (int i = 0; i < n; i++)
        printf("%.2f ", arr[i]);
    printf("\n");
}

void displayTopFive(float arr[], int n) {
    printf("Top 5 Scores:\n");
    int count = n < 5 ? n : 5;
    for (int i = n - 1; i >= n - count; i--)
        printf("%.2f ", arr[i]);
```

```

    printf("\n");
}

// ===== MAIN =====
int main() {
    float arr[100];
    int n, choice;

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

    printf("Enter percentages of students:\n");
    for (int i = 0; i < n; i++)
        scanf("%f", &arr[i]);

    do {
        printf("\n--- Sorting Program --\n");
        printf("1. Selection Sort\n");
        printf("2. Bubble Sort\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                selectionSort(arr, n);
                printf("Array sorted using Selection Sort.\n");
                display(arr, n);
                displayTopFive(arr, n);
                break;

            case 2:
                bubbleSort(arr, n);
                printf("Array sorted using Bubble Sort.\n");
                display(arr, n);
                displayTopFive(arr, n);
                break;

            case 0:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 0);

    return 0;
}

```

```
Enter the number of students: 5
Enter the percentage of 5 students:
70 85 67 90 69
```

```
Menu:
1. Selection Sort
2. Bubble Sort
Enter your choice: 1
```

```
Array sorted using Selection Sort.
Sorted percentages:
67.00 69.00 70.00 85.00 90.00
Top 5 scores:
90.00 85.00 70.00 69.00 67.00
```

```
==== Code Execution Successful ===
```

Tab 3

3. Consider the telephone book database of Nclients. Make use of a hash table implementation to quickly look up a client's telephone number. Make use of linear probing, double hashing and quadratic collision handling techniques.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 20 // size of hash table
// Structure for storing client info
typedef struct {
char name[30];
char phone[15];
int occupied; // 0 = empty, 1 = occupied
} Client;
Client hashTable[MAX];
// ===== HASH FUNCTIONS =====
// Primary hash function
int hash1(char *key) {
int sum = 0;
for (int i = 0; key[i] != '\0'; i++) {
sum += key[i];
}
return sum % MAX;
}
// Secondary hash function (for double hashing)
int hash2(char *key) {
int sum = 0;
for (int i = 0; key[i] != '\0'; i++) {
sum += key[i];
}
return (7 - (sum % 7)); // must be non-zero
}
// ===== INSERT FUNCTIONS =====
// Linear Probing
void insertLinear(char *name, char *phone) {
int index = hash1(name);
int i = 0;
while (hashTable[(index + i) % MAX].occupied) {
i++;
if (i == MAX) {
printf("Hash Table Full!\n");
return;
}
}
strcpy(hashTable[(index + i) % MAX].name, name);
strcpy(hashTable[(index + i) % MAX].phone, phone);
hashTable[(index + i) % MAX].occupied = 1;
}
// Quadratic Probing
void insertQuadratic(char *name, char *phone) {
int index = hash1(name);
int i = 0;
while (hashTable[(index + i * i) % MAX].occupied) {
```

```

i++;
if (i == MAX) {
printf("Hash Table Full!\n");
return;
}
strcpy(hashTable[(index + i * i) % MAX].name, name);
strcpy(hashTable[(index + i * i) % MAX].phone, phone);
hashTable[(index + i * i) % MAX].occupied = 1;

}

// Double Hashing
void insertDouble(char *name, char *phone) {
int index1 = hash1(name);
int index2 = hash2(name);
int i = 0;
while (hashTable[(index1 + i * index2) % MAX].occupied) {
i++;
if (i == MAX) {
printf("Hash Table Full!\n");
return;
}
strcpy(hashTable[(index1 + i * index2) % MAX].name, name);
strcpy(hashTable[(index1 + i * index2) % MAX].phone, phone);
hashTable[(index1 + i * index2) % MAX].occupied = 1;
}

// ===== SEARCH FUNCTIONS =====
// Linear Probing Search
void searchLinear(char *name) {
int index = hash1(name);
int i = 0;
while (hashTable[(index + i) % MAX].occupied) {
if (strcmp(hashTable[(index + i) % MAX].name, name) == 0) {
printf("Found: %s -> %s\n", name, hashTable[(index + i) % MAX].phone);
return;
}
i++;
if (i == MAX) break;
}
printf("Client not found!\n");

}

// Quadratic Probing Search
void searchQuadratic(char *name) {
int index = hash1(name);
int i = 0;
while (hashTable[(index + i * i) % MAX].occupied) {
if (strcmp(hashTable[(index + i * i) % MAX].name, name) == 0) {
printf("Found: %s -> %s\n", name, hashTable[(index + i * i) % MAX].phone);
return;
}
i++;
if (i == MAX) break;
}
}

```

```

}

printf("Client not found!\n");
}

// Double Hashing Search
void searchDouble(char *name) {
int index1 = hash1(name);
int index2 = hash2(name);
int i = 0;
while (hashTable[(index1 + i * index2) % MAX].occupied) {
if (strcmp(hashTable[(index1 + i * index2) % MAX].name, name) == 0) {
printf("Found: %s -> %s\n", name, hashTable[(index1 + i * index2) % MAX].phone);
return;
}
i++;
if (i == MAX) break;
}
printf("Client not found!\n");
}

// ===== DISPLAY =====

void displayTable() {
printf("\nHash Table:\n");
for (int i = 0; i < MAX; i++) {
if (hashTable[i].occupied)
printf("[%d] %s -> %s\n", i, hashTable[i].name, hashTable[i].phone);
else
printf("[%d] ---\n", i);
}
}

// ===== MAIN =====

int main() {
int choice;
char name[30], phone[15];
// initialize table
for (int i = 0; i < MAX; i++) {
hashTable[i].occupied = 0;
}
do {
printf("\n--- Telephone Book using Hash Table ---\n");
printf("1. Insert (Linear Probing)\n");
printf("2. Insert (Quadratic Probing)\n");
printf("3. Insert (Double Hashing)\n");
printf("4. Search (Linear Probing)\n");
printf("5. Search (Quadratic Probing)\n");
printf("6. Search (Double Hashing)\n");
printf("7. Display Table\n");
printf("0. Exit\n");
printf("Enter choice: ");
scanf("%d", &choice);
switch(choice) {
case 1:

printf("Enter name: ");
scanf("%s", name);

```

```
printf("Enter phone: ");
scanf("%s", phone);
insertLinear(name, phone);
break;
case 2:
printf("Enter name: ");
scanf("%s", name);
printf("Enter phone: ");
scanf("%s", phone);
insertQuadratic(name, phone);
break;
case 3:
printf("Enter name: ");
scanf("%s", name);
printf("Enter phone: ");
scanf("%s", phone);
insertDouble(name, phone);
break;
case 4:
printf("Enter name to search: ");
scanf("%s", name);
searchLinear(name);
break;
case 5:
printf("Enter name to search: ");
scanf("%s", name);
searchQuadratic(name);
break;
case 6:
printf("Enter name to search: ");
scanf("%s", name);
searchDouble(name);
break;
case 7:
displayTable();
break;
case 0:
printf("Exiting...\n");
break;
default:
printf("Invalid choice!\n");
}
} while(choice != 0);
return 0;
}
```

```
--- Telephone Book using Hash Table ---
```

1. Insert (Linear Probing)
2. Insert (Quadratic Probing)
3. Insert (Double Hashing)
4. Search (Linear Probing)
5. Search (Quadratic Probing)
6. Search (Double Hashing)
7. Display Table

```
0. Exit
```

```
Enter choice: 2
```

```
Enter name: Aarvee
```

```
Enter phone: 76324
```

```
--- Telephone Book using Hash Table ---
```

1. Insert (Linear Probing)
2. Insert (Quadratic Probing)
3. Insert (Double Hashing)
4. Search (Linear Probing)
5. Search (Quadratic Probing)
6. Search (Double Hashing)
7. Display Table

```
0. Exit
```

```
Enter choice: 7
```

```
Hash Table:
```

```
[0] ---
```

```
[1] ---
```

```
[2] ---
```

Tab 4

4. To design and implement a C program using Singly Linked List (SLL) to maintain the membership records of the Pinnacle Club, including operations like insertion, deletion, counting, display, and concatenation of lists.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NAME_LEN 50
typedef struct Node {
    int prn;
    char name[NAME_LEN];
    struct Node* next;
} Node;
// Function prototypes
Node* createNode(int prn, const char* name);
void addPresident(Node** head, int prn, const char* name);
void addSecretary(Node** head, int prn, const char* name);
void addMember(Node** head, int prn, const char* name);
void deletePresident(Node** head);
void deleteSecretary(Node** head);
void deleteMember(Node** head, int prn);
void displayMembers(Node* head);
int countMembers(Node* head);

Node* concatenate(Node* head1, Node* head2);
// Create new node
Node* createNode(int prn, const char* name) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->prn = prn;
    strncpy(newNode->name, name, NAME_LEN);
    newNode->next = NULL;
    return newNode;
}
// Add President
void addPresident(Node** head, int prn, const char* name) {
    Node* newNode = createNode(prn, name);
    newNode->next = *head;
    *head = newNode;
}
// Add Secretary
void addSecretary(Node** head, int prn, const char* name) {
    Node* newNode = createNode(prn, name);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}
// Add Member (in between President and Secretary)
void addMember(Node** head, int prn, const char* name) {
    Node* newNode = createNode(prn, name);
    if (*head == NULL || (*head)->next == NULL) {
```

```

printf("Add President and Secretary first.\n");
return;
}
Node* temp = *head;
while (temp->next->next != NULL)

temp = temp->next;
newNode->next = temp->next;
temp->next = newNode;
}

// Delete President
void deletePresident(Node** head) {
if (*head == NULL) {
printf("List is empty.\n");
return;
}
Node* temp = *head;
*head = (*head)->next;
free(temp);
}

// Delete Secretary
void deleteSecretary(Node** head) {
if (*head == NULL || (*head)->next == NULL) {
printf("Not enough members to delete Secretary.\n");
return;
}
Node* temp = *head;
while (temp->next->next != NULL)
temp = temp->next;
free(temp->next);
temp->next = NULL;
}

// Delete Member by PRN
void deleteMember(Node** head, int prn) {
if (*head == NULL) {
printf("List is empty.\n");
return;
}
Node* temp = *head;
Node* prev = NULL;
// Cannot delete President or Secretary here

if (temp->prn == prn) {
printf("Use deletePresident() for President.\n");
return;
}
while (temp != NULL && temp->prn != prn) {
prev = temp;
temp = temp->next;
}
if (temp == NULL || temp->next == NULL) {
printf("Cannot delete Secretary or PRN not found.\n");
return;
}

```

```

prev->next = temp->next;
free(temp);
}
// Display Members
void displayMembers(Node* head) {
if (head == NULL) {
printf("Club is empty.\n");
return;
}
Node* temp = head;
printf("\nClub Members:\n");
while (temp != NULL) {
printf("PRN: %d, Name: %s\n", temp->prn, temp->name);
temp = temp->next;
}
}
// Count Members
int countMembers(Node* head) {
int count = 0;
Node* temp = head;
while (temp != NULL) {
count++;
temp = temp->next;
}
return count;
}
// Concatenate two club lists
Node* concatenate(Node* head1, Node* head2) {
if (head1 == NULL) return head2;
Node* temp = head1;
while (temp->next != NULL)
temp = temp->next;
temp->next = head2;
return head1;
}
// Main menu
int main() {
Node* club1 = NULL;
Node* club2 = NULL;
int choice, prn;
char name[NAME_LEN];
do {
printf("\n--- Pinnacle Club Menu ---\n");
printf("1. Add President\n2. Add Member\n3. Add Secretary\n");
printf("4. Delete President\n5. Delete Member\n6. Delete Secretary\n");
printf("7. Display Members\n8. Count Members\n");
printf("9. Create second division and concatenate\n10. Exit\n");
printf("Enter choice: ");
scanf("%d", &choice);
getchar(); // To clear buffer
switch (choice) {
case 1:
printf("Enter PRN and Name for President: ");

```

```

scanf("%d", &prn);
getchar();
fgets(name, NAME_LEN, stdin);
name[strcspn(name, "\n")] = '\0';
addPresident(&club1, prn, name);
break;

case 2:
printf("Enter PRN and Name for Member: ");
scanf("%d", &prn);
getchar();
fgets(name, NAME_LEN, stdin);
name[strcspn(name, "\n")] = '\0';
addMember(&club1, prn, name);
break;
case 3:
printf("Enter PRN and Name for Secretary: ");
scanf("%d", &prn);
getchar();
fgets(name, NAME_LEN, stdin);
name[strcspn(name, "\n")] = '\0';
addSecretary(&club1, prn, name);
break;
case 4:
deletePresident(&club1);
break;
case 5:
printf("Enter PRN to delete: ");
scanf("%d", &prn);
deleteMember(&club1, prn);
break;
case 6:
deleteSecretary(&club1);
break;
case 7:
displayMembers(club1);
break;
case 8:
printf("Total Members: %d\n", countMembers(club1));
break;
case 9:
// Create a second division (club2)
printf("Creating second club (division) with 2 members...\n");
addPresident(&club2, 201, "Div2_President");
addMember(&club2, 202, "Div2_Member");
addSecretary(&club2, 203, "Div2_Secretary");
printf("Second club:\n");

displayMembers(club2);
// Concatenate
club1 = concatenate(club1, club2);
printf("After concatenation:\n");
displayMembers(club1);
break;

```

```
case 10:  
printf("Exiting...\n");  
break;  
default:  
printf("Invalid choice.\n");  
}  
} while (choice != 10);  
return 0;  
}
```

```
--- Pinnacle Club Menu ---  
1. Add President  
2. Add Member  
3. Add Secretary  
4. Delete President  
5. Delete Member  
6. Delete Secretary  
7. Display Members  
8. Count Members  
9. Create second division and concatenate  
10. Exit  
Enter choice: 1  
Enter PRN and Name for President: 3524 Aarvee  
  
--- Pinnacle Club Menu ---  
1. Add President  
2. Add Member  
3. Add Secretary  
4. Delete President  
5. Delete Member  
6. Delete Secretary  
7. Display Members  
8. Count Members  
9. Create second division and concatenate  
10. Exit  
Enter choice: |
```

Tab 5

5. To implement a ticket booking system for Cinemax Theatre using a doubly linked list for efficient management of available and booked seats.

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

#define ROWS 10
#define SEATS 7

// ===== STRUCTURE =====
typedef struct Seat {
    int seatNo;          // Seat number in the row
    int isBooked;        // 0 = Free, 1 = Booked
    struct Seat *prev;
    struct Seat *next;
} Seat;

// ===== GLOBAL ARRAY =====
Seat *rowHeads[ROWS]; // Array of head pointers for each row

// ===== CREATE A ROW =====
Seat* createRow() {
    Seat *head = NULL, *temp = NULL, *newNode;

    for (int i = 1; i <= SEATS; i++) {
        newNode = (Seat*)malloc(sizeof(Seat));
        newNode->seatNo = i;
        newNode->isBooked = 0;
        newNode->next = NULL;
        newNode->prev = temp;

        if (temp)
            temp->next = newNode;
        else
            head = newNode;

        temp = newNode;
    }
    return head;
}

// ===== INITIALIZE THEATER =====
void initTheater() {
    for (int i = 0; i < ROWS; i++) {
        rowHeads[i] = createRow();
    }
}

// ===== DISPLAY SEATING LAYOUT =====
void displayHall() {
    printf("\n--- Cinemax Theater Seating Layout ---\n");
    printf("Legend: [O] Free [X] Booked\n\n");

    for (int i = 0; i < ROWS; i++) {
```

```

Seat *temp = rowHeads[i];
printf("Row %2d: ", i + 1);
while (temp) {
    printf("[%c] ", temp->isBooked ? 'X' : 'O');
    temp = temp->next;
}
printf("\n");
}

// ===== BOOK A SEAT =====
void bookSeat(int row, int seatNo) {
    Seat *temp = rowHeads[row];
    while (temp && temp->seatNo != seatNo)
        temp = temp->next;

    if (!temp) {
        printf("Invalid seat number!\n");
        return;
    }

    if (temp->isBooked) {
        printf("Seat already booked!\n");
    } else {
        temp->isBooked = 1;
        printf("✓ Seat %d in Row %d booked successfully.\n", seatNo, row + 1);
    }
}

// ===== CANCEL A BOOKING =====
void cancelSeat(int row, int seatNo) {
    Seat *temp = rowHeads[row];
    while (temp && temp->seatNo != seatNo)
        temp = temp->next;

    if (!temp) {
        printf("Invalid seat number!\n");
        return;
    }

    if (!temp->isBooked) {
        printf("Seat is not booked!\n");
    } else {
        temp->isBooked = 0;
        printf("✗ Booking of Seat %d in Row %d cancelled.\n", seatNo, row + 1);
    }
}

// ===== MAIN =====
int main() {
    int choice, row, seat;

    initTheater(); // Initialize the theater layout
}

```

```

// Pre-book some seats
bookSeat(0, 3);
bookSeat(2, 5);
bookSeat(5, 7);

while (1) {
    printf("\n--- Cinemax Theater Booking System ---\n");
    printf("1. Show seating layout\n");
    printf("2. Book a seat\n");
    printf("3. Cancel booking\n");
    printf("4. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            displayHall();
            break;

        case 2:
            printf("Enter row (1-10) and seat (1-7): ");
            scanf("%d %d", &row, &seat);
            if (row >= 1 && row <= ROWS && seat >= 1 && seat <= SEATS)
                bookSeat(row - 1, seat);
            else
                printf("Invalid row or seat number!\n");
            break;

        case 3:
            printf("Enter row (1-10) and seat (1-7): ");
            scanf("%d %d", &row, &seat);
            if (row >= 1 && row <= ROWS && seat >= 1 && seat <= SEATS)
                cancelSeat(row - 1, seat);
            else
                printf("Invalid row or seat number!\n");
            break;

        case 4:
            printf("Exiting... Thank you!\n");
            return 0;

        default:
            printf("Invalid choice! Try again.\n");
    }
}

return 0;
}

```

```
 Seat 3 in Row 1 booked successfully.  
 Seat 5 in Row 3 booked successfully.  
 Seat 7 in Row 6 booked successfully.
```

```
--- Cinemax Theater Booking System ---
```

1. Show seating layout
2. Book a seat
3. Cancel booking
4. Exit

```
Enter choice: 3
```

```
Enter row (1-10) and seat (1-7): 7 6
```

```
Seat is not booked!
```

```
--- Cinemax Theater Booking System ---
```

1. Show seating layout
2. Book a seat
3. Cancel booking
4. Exit

```
Enter choice:
```

Tab 6

6. To implement a program for expression conversion from infix to postfix and evaluate the postfix expression using stack, considering operands and operators as single characters and only +, -, and * operators.

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

#define MAX 100

char opStack[MAX];
int opTop = -1;

int valStack[MAX];
int valTop = -1;

// Push to operator stack
void pushOp(char ch) {
    opStack[++opTop] = ch;
}

char popOp() {
    return opStack[opTop--];
}

char peekOp() {
    return opStack[opTop];
}

// Push to value stack
void pushVal(int val) {
    valStack[++valTop] = val;
}

int popVal() {
    return valStack[valTop--];
}

// Check if operator

int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Get precedence
int precedence(char ch) {
    if (ch == '*' || ch == '/') return 2;
    if (ch == '+' || ch == '-') return 1;
    return 0;
}

// Convert infix to postfix
void infixToPostfix(char* infix, char* postfix) {
    int i = 0, j = 0;
    char ch;
```

```

while ((ch = infix[i++]) != '\0') {
if (isalpha(ch)) {
postfix[j++] = ch;
}
else if (isOperator(ch)) {
while (opTop != -1 && precedence(peekOp()) >= precedence(ch)) {
postfix[j++] = popOp();
}
pushOp(ch);
}
}

while (opTop != -1) {
postfix[j++] = popOp();
}

postfix[j] = '\0';
}

// Evaluate postfix
int evaluatePostfix(char* postfix) {
int i = 0;
char ch;
int values[26] = {0};

// Get values for variables
for (i = 0; postfix[i]; i++) {
ch = postfix[i];
if (isalpha(ch) && values[ch - 'a'] == 0) {
printf("Enter value of %c: ", ch);
scanf("%d", &values[ch - 'a']);
}
}

i = 0;
while ((ch = postfix[i++]) != '\0') {
if (isalpha(ch)) {
pushVal(values[ch - 'a']);
}
else if (isOperator(ch)) {
int b = popVal();
int a = popVal();
switch (ch) {
case '+': pushVal(a + b); break;
case '-': pushVal(a - b); break;
case '*': pushVal(a * b); break;
case '/': pushVal(a / b); break;
}
}
}

return popVal();
}

```

```
}

int main() {
char infix[MAX], postfix[MAX];

printf("Enter infix expression (e.g. a+b*c): ");
scanf("%s", infix);

infixToPostfix(infix, postfix);
printf("Postfix: %s\n", postfix);

int result = evaluatePostfix(postfix);
printf("Result: %d\n", result);

return 0;
}
```

```
Enter infix expression (e.g. a+b*c): a+b
Postfix: ab+
Enter value of a: 3
Enter value of b: 4
Result: 7
```

```
==== Code Execution Successful ===
```

Tab 7

7. To write a program in C that simulates a job queue (representing a pizza parlor system) where orders are added and served in a First Come First Served (FCFS) manner, with a maximum capacity of M orders.

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

#define MAX 100

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

// Function to add a job to the queue
void addJob(int job) {
    if (rear == MAX - 1) {
        printf("Queue Overflow. Cannot add more jobs.\n");
        return;
    }
    if (front == -1) front = 0;
    queue[++rear] = job;
    printf("Job %d added to the queue.\n", job);
}

// Function to delete a job from the queue
void deleteJob() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow. No jobs to delete.\n");
        return;
    }
    int job = queue[front++];
    printf("Job %d deleted from the queue.\n", job);

    // Reset the queue if it becomes empty
    if (front > rear)
        front = rear = -1;
}

// Function to display the queue
void displayQueue() {
    if (front == -1) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Current Jobs in Queue: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

// Main function
int main() {
    int choice, job;
```

```
while (1) {
    printf("\n--- Job Scheduling using Queue ---\n");
    printf("1. Add Job\n");
    printf("2. Delete Job\n");
    printf("3. Display Queue\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter job number to add: ");
            scanf("%d", &job);
            addJob(job);
            break;

        case 2:
            deleteJob();
            break;

        case 3:
            displayQueue();
            break;

        case 4:
            printf("Exiting...\n");
            exit(0);

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

return 0;
}
```

```
--- Job Scheduling using Queue ---
1. Add Job
2. Delete Job
3. Display Queue
4. Exit
Enter your choice: 1
Enter job number to add: 23
Job 23 added to the queue.

--- Job Scheduling using Queue ---
1. Add Job
2. Delete Job
3. Display Queue
4. Exit
Enter your choice: 3
Current Jobs in Queue: 23

--- Job Scheduling using Queue ---
1. Add Job
2. Delete Job
3. Display Queue
4. Exit
Enter your choice:
```

Tab 8

8. Implementation of Binary Search Tree (BST) and its operations.

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

// Structure for a tree node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert into BST
Node* insert(Node* root, int data) {
    if (root == NULL)
        return createNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}

// Inorder Traversal
void inorder(Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Find height of the tree
int height(Node* root) {
    if (root == NULL)
        return 0;
    int leftHeight = height(root->left);
    int rightHeight = height(root->right);
    return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
}

// Find minimum value
int findMin(Node* root) {
    if (root == NULL) {
        printf("Tree is empty.\n");
        return -1;
    }
```

```

}

Node* current = root;
while (current->left != NULL)
    current = current->left;
return current->data;
}

// Search for a value
int search(Node* root, int key) {
    if (root == NULL)
        return 0;
    if (root->data == key)
        return 1;
    else if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}

// Mirror the tree
void mirror(Node* root) {
    if (root == NULL)
        return;
    mirror(root->left);
    mirror(root->right);

    Node* temp = root->left;
    root->left = root->right;
    root->right = temp;
}

// ====== MAIN ======
int main() {
    Node* root = NULL;
    int choice, value;

    while (1) {
        printf("\n--- Binary Search Tree Operations ---\n");
        printf("1. Insert\n");
        printf("2. Inorder Traversal\n");
        printf("3. Search\n");
        printf("4. Find Height\n");
        printf("5. Find Minimum Value\n");
        printf("6. Mirror Tree\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
        }
    }
}

```

```

case 2:
    printf("Inorder Traversal: ");
    inorder(root);
    printf("\n");
    break;

case 3:
    printf("Enter value to search: ");
    scanf("%d", &value);
    if (search(root, value))
        printf("%d found in BST.\n", value);
    else
        printf("%d not found in BST.\n", value);
    break;

case 4:
    printf("Height of BST: %d\n", height(root));
    break;

case 5:
    value = findMin(root);
    if (value != -1)
        printf("Minimum value in BST: %d\n", value);
    break;

case 6:
    mirror(root);
    printf("Tree has been mirrored.\n");
    break;

case 7:
    printf("Exiting program.\n");
    exit(0);

default:
    printf("Invalid choice. Try again.\n");
}

}

return 0;
}

```

```
--- Binary Search Tree Operations ---
1. Insert
2. Inorder Traversal
3. Search
4. Find Height
5. Find Minimum Value
6. Mirror Tree
7. Exit
Enter your choice: 1
Enter value to insert: 21

--- Binary Search Tree Operations ---
1. Insert
2. Inorder Traversal
3. Search
4. Find Height
5. Find Minimum Value
6. Mirror Tree
7. Exit
Enter your choice: 4
Height of BST: 1

--- Binary Search Tree Operations ---
1. Insert
2. Inorder Traversal
3. Search
4. Find Height
5. Find Minimum Value
6. Mirror Tree
7. Exit
Enter your choice: |
```

Tab 9

9. To represent flight paths between cities using a graph data structure with adjacency matrix or adjacency list representation, and to check whether the graph is connected or not.

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

#define MAX 100

// Structure for adjacency list node
typedef struct Node {
    int dest;
    int cost;
    struct Node* next;
} Node;

// Array of city names
char cities[MAX][100];
int cityCount = 0;

// Adjacency list array
Node* adjList[MAX];

// Visited array for BFS
int visited[MAX];

// Function to add a city
void addCity(char name[]) {
    strcpy(cities[cityCount], name);
    adjList[cityCount] = NULL;
    cityCount++;
}

// Find index of city by name
int getCityIndex(char name[]) {
    for (int i = 0; i < cityCount; i++) {
        if (strcmp(cities[i], name) == 0)
            return i;
    }
    return -1;
}

// Add edge (flight path)
void addFlight(char city1[], char city2[], int cost) {
    int i = getCityIndex(city1);
    int j = getCityIndex(city2);

    if (i == -1 || j == -1) {
        printf("Invalid city name(s).\n");
        return;
    }

    // Add city2 to city1's list
```

```

Node* newNode = (Node*)malloc(sizeof(Node));
newNode->dest = j;
newNode->cost = cost;
newNode->next = adjList[i];
adjList[i] = newNode;

// Since it's an undirected graph, add city1 to city2's list
newNode = (Node*)malloc(sizeof(Node));
newNode->dest = i;
newNode->cost = cost;
newNode->next = adjList[j];
adjList[j] = newNode;
}

// Display the graph
void displayGraph() {
    printf("\n--- Flight Graph --\n");
    for (int i = 0; i < cityCount; i++) {
        printf("%s -> ", cities[i]);
        Node* temp = adjList[i];
        while (temp) {
            printf("[%s (Cost: %d)] -> ", cities[temp->dest], temp->cost);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

// BFS to check connectivity
void BFS(int start) {
    int queue[MAX], front = 0, rear = 0;
    queue[rear++] = start;
    visited[start] = 1;

    while (front < rear) {
        int current = queue[front++];
        Node* temp = adjList[current];
        while (temp) {
            if (!visited[temp->dest]) {
                queue[rear++] = temp->dest;
                visited[temp->dest] = 1;
            }
            temp = temp->next;
        }
    }
}

// Check if graph is connected
void checkConnectivity() {
    for (int i = 0; i < cityCount; i++)
        visited[i] = 0;

    BFS(0); // Start from first city
}

```

```

for (int i = 0; i < cityCount; i++) {
    if (!visited[i]) {
        printf("Graph is NOT connected. '%s' is not reachable.\n", cities[i]);
        return;
    }
}
printf("Graph is connected.\n");
}

// MAIN MENU
int main() {
    int choice, cost;
    char city1[100], city2[100];

    while (1) {
        printf("\n--- Flight Path Graph Menu ---\n");
        printf("1. Add city\n");
        printf("2. Add flight path\n");
        printf("3. Display flight graph\n");
        printf("4. Check if graph is connected\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Consume newline

        switch (choice) {
            case 1:
                printf("Enter city name: ");
                fgets(city1, sizeof(city1), stdin);
                city1[strcspn(city1, "\n")] = 0; // Remove newline
                addCity(city1);
                break;

            case 2:
                printf("Enter source city: ");
                fgets(city1, sizeof(city1), stdin);
                city1[strcspn(city1, "\n")] = 0;
                printf("Enter destination city: ");
                fgets(city2, sizeof(city2), stdin);
                city2[strcspn(city2, "\n")] = 0;
                printf("Enter cost (time/fuel): ");
                scanf("%d", &cost);
                getchar();
                addFlight(city1, city2, cost);
                break;

            case 3:
                displayGraph();
                break;

            case 4:
                if (cityCount == 0) {
                    printf("No cities added.\n");
                } else {

```

```

        checkConnectivity();
    }
    break;

case 5:
    printf("Exiting...\n");
    return 0;

default:
    printf("Invalid choice.\n");
}
}

return 0;
}

```

```

--- Flight Path Graph Menu ---
1. Add city
2. Add flight path
3. Display flight graph
4. Check if graph is connected
5. Exit
Enter your choice: 1
Enter city name: pune

--- Flight Path Graph Menu ---
1. Add city
2. Add flight path
3. Display flight graph
4. Check if graph is connected
5. Exit
Enter your choice: 2
Enter source city: pune
Enter destination city: mumbai
Enter cost (time/fuel): 120
Invalid city name(s).

--- Flight Path Graph Menu ---
1. Add city
2. Add flight path
3. Display flight graph
4. Check if graph is connected
5. Exit
Enter your choice: |

```