HEMCHAND YADAV UNIVERSITY

DURG, CHHATTISGARH

KALYAN PG COLLEGE SECTOR-7 ,BHILAI

SESSION 2024-25

DATA STRUCTURE THROUGH ALGORITHMS USING **"C"**

HEAD OF DEPARTMENT

MRS. PRANALI HATWAR

GUIDED BY
MR. ABHINAV SIR

PRESENTED BY
SAMEER BANCHHOR

INDEX

PRACTICAL 1

Write an algorithm and program to insert an element in an array.

ALGORITHM :

1. **Start**
2. **Declare**: `arr[100], n, pos, i, val`
3. **Input**: Read `n` (number of elements)
4. **Input Array**: For `i = 0` to `n - 1`, read `arr[i]`
5. **Input**: Read `pos` and `val` (position and value to insert)
6. **Shift Elements**: For `i = n` down to `pos`, set `arr[i] = arr[i - 1]`
7. **Insert Value**: Set `arr[pos - 1] = val`
8. **Increment**: `n++`
9. **Output**: Print updated array `arr[0]` to `arr[n - 1]`
10. **End**

CODE:

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

void main() {
    int arr[100], n, pos, i, val;
    clrscr();

    printf("Enter number of elements: ");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
```

```
        scanf("%d", &arr[i]);

    printf("Enter position and value to insert: ");
    scanf("%d %d", &pos, &val);

    for(i = n; i >= pos; i--)
        arr[i] = arr[i - 1];

    arr[pos - 1] = val;
    n++;

    printf("Array after insertion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);

    getch();
}
```

Output:

```
Enter number of elements: 5
10
20
30
40
50
Enter position and value to insert: 3 25
Array after insertion: 10 20 25 30 40 50
```

PRACTICAL 2

Write an algorithm and program to delete an element from an array.

Algorithm:

1. **Start**
2. **Declare**: `arr[100], n, pos, i`
3. **Clear Screen** (optional, specific to certain compilers)
4. **Input**: Read `n` (number of elements)
5. **Input Array**: For `i = 0` to `n - 1`, read `arr[i]`
6. **Input**: Read `pos` (position to delete)
7. **Shift Elements**: For `i = pos - 1` to `n - 2`, set `arr[i] = arr[i + 1]`
8. **Decrement Size**: `n--`
9. **Output**: Print updated array `arr[0]` to `arr[n - 1]`
10. **End**

CODE:

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

void main() {
    int arr[100], n, pos, i;
    clrscr();
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter position to delete: ");
    scanf("%d", &pos);

    for(i = pos - 1; i < n - 1; i++)
        arr[i] = arr[i + 1];
    n--;
    printf("Array after deletion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);

    getch();
}
```

Output:

```
Enter number of elements: 5
Enter the elements:
10
20
30
40
50
Enter position to delete: 3
Array after deletion: 10 20 40 50
```

PRACTICAL 3

Write an algorithm and program to add two matrix A and B.

algorithm:

1. **Declare Variables**: Initialize matrices `A`, `B`, `C` and variables `rows`, `cols`, `i`, `j`.
2. **Clear Screen**: Call `clrscr()`.
3. **Input Dimensions**: Prompt and read `rows` and `cols`.
4. **Input Matrix A**: Prompt and read elements into matrix `A`.
5. **Input Matrix B**: Prompt and read elements into matrix `B`.
6. **Add Matrices**: Compute `C[i][j] = A[i][j] + B[i][j]` for all elements.
7. **Output Matrix C**: Print the resultant matrix `C`.
8. **Wait for Input**: Call `getch()`.

code:

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

void main() {
    int A[10][10], B[10][10], C[10][10], rows, cols, i, j;
    clrscr();
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
    printf("Enter elements of matrix A:\n");
    for(i = 0; i < rows; i++)
        for(j = 0; j < cols; j++)
            scanf("%d", &A[i][j]);
```

```
    printf("Enter elements of matrix B:\n");
    for(i = 0; i < rows; i++)
        for(j = 0; j < cols; j++)
            scanf("%d", &B[i][j]);

    for(i = 0; i < rows; i++)
        for(j = 0; j < cols; j++)
            C[i][j] = A[i][j] + B[i][j];

    printf("Resultant matrix C (A + B):\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++)
            printf("%d ", C[i][j]);
        printf("\n");
    }

    getch();
}
```

output:

```
Enter number of rows and columns: 2 2
Enter elements of matrix A:
1 2
3 4
Enter elements of matrix B:
5 6
7 8
Resultant matrix C (A + B):
6 8
10 12
```

PRACTICAL 4

Write an algorithm and program to multiply two matrix A and B.

algorithm:

1. **Input**: Accept elements for two 2x2 matrices A and B.
2. **Initialize**: Set matrix C to 0.
3. **Multiply**: For each element $C[i][j]$, compute:

$$C[i][j] = A[i][0] * B[0][j] + A[i][1] * B[1][j]$$

4. **Display**: Output the resulting matrix C.

code:

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

void main() {
    int A[2][2], B[2][2], C[2][2], i, j, k;
    clrscr();

    printf("Enter elements of matrix A (2x2):\n");
    for(i = 0; i < 2; i++)
        for(j = 0; j < 2; j++)
            scanf("%d", &A[i][j]);

    printf("Enter elements of matrix B (2x2):\n");
    for(i = 0; i < 2; i++)
        for(j = 0; j < 2; j++)
```

```
            scanf("%d", &B[i][j]);

    for(i = 0; i < 2; i++)
        for(j = 0; j < 2; j++) {
            C[i][j] = 0;
            for(k = 0; k < 2; k++)
                C[i][j] += A[i][k] * B[k][j];
        }

    printf("Resultant matrix C (A * B):\n");
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++)
            printf("%d ", C[i][j]);
        printf("\n");
    }

    getch();
}
```

OUTPUT :

```
Enter elements of matrix A (2x2):
1 2
3 4
Enter elements of matrix B (2x2):
2 0
1 2
Resultant matrix C (A * B):
4 4
10 8
```

PRACTICAL 5

Write an algorithm and program to Implementation of linked list using array.

ALGORITHM:

1. **Initialize Stack**:
   - Define an array `stack[MAX]` to store elements.
   - Set `top = -1` to indicate that the stack is initially empty.
2. **Push Operation**:
   - If `top` is equal to `MAX - 1`, print "Stack Overflow".
   - Otherwise, increment `top` and insert the element into `stack[top]`.
3. **Pop Operation**:
   - If `top` is `-1`, print "Stack Underflow".
   - Otherwise, return and remove the element at `stack[top]`, and decrement `top`.
4. **Display Operation**:
   - If the stack is empty (i.e., `top == -1`), print "Stack is empty".
   - Otherwise, print all elements from `stack[top]` down to `stack[0]`.
5. **Menu**:
   - Continuously prompt the user for an operation until they choose to exit.

CODE :

```
#include <stdio.h>
#include <conio.h>
#define MAX 5

int stack[MAX], top = -1;

void push(int val) {
```

```
    if(top == MAX - 1)
        printf("Stack Overflow\n");
    else
        stack[++top] = val;
}

int pop() {
    if(top == -1) {
        printf("Stack Underflow\n");
        return -1;
    } else
        return stack[top--];
}

void display() {
    int i;
    if(top == -1)
        printf("Stack is empty\n");
    else {
        printf("Stack elements: ");
        for(i = top; i >= 0; i--)
            printf("%d ", stack[i]);
        printf("\n");
    }
}

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

    do {
```

```
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter value to push: ");
                    scanf("%d", &val);
                    push(val);
                    break;
            case 2: val = pop();
                    if(val != -1) printf("Popped value: %d\n", val);
                    break;
            case 3: display();
                    break;
        }
    } while(choice != 4);

    getch();
}
```

OUTPUT :

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
```

```
Enter value to push: 20

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack elements: 20 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Popped value: 20

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack elements: 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
```

PRACTICAL 6

Write an algorithm and program to insert an item into double linked list.

ALGORITHM:

1. **Define Node Structure**:
   - Create a `Node` structure with three fields: `data`, `prev`, and `next`.
2. **Insert at Front**:
   - Create a new node with the given value.
   - Set the `prev` pointer of the new node to `NULL`.
   - Set the `next` pointer of the new node to point to the current head of the list.
   - If the list is not empty, update the `prev` pointer of the current head to the new node.
   - Update the head pointer to the new node.
3. **Display List**:
   - Traverse the list starting from the head, printing each node's `data` value.
4. **Main Program**:
   - Start with an empty list (`head = NULL`).
   - Insert several nodes at the front.
   - Display the final list.

CODE:

```
// Write an algorithm and program to insert an item into double linked list. 6


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

struct Node {
```

```
    int data;
    struct Node *prev, *next;
};

void insertFront(struct Node **head, int val) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->prev = NULL;
    newNode->next = *head;
    if(*head)
        (*head)->prev = newNode;
    *head = newNode;
}

void display(struct Node *node) {
    while(node) {
        printf("%d ", node->data);
        node = node->next;
    }
}

void main() {
    struct Node *head = NULL;
    clrscr();

    insertFront(&head, 10);
    insertFront(&head, 20);
    insertFront(&head, 30);

    printf("Doubly linked list: ");
    display(head);
```

```
    getch();
}
```

OUTPUT :

```
Doubly linked list: 30 20 10
```

PRACTICAL 7

Write an algorithm and program to delete an item from double linked list.

ALGORITHM:

1. **Define Node Structure**:
   - Define a structure `Node` with fields: `data`, `prev`, and `next`.
2. **Delete Node**:
   - Start with the head of the list.
   - Traverse the list until the node with the given value (`key`) is found.
   - If the node is found:
     - If the node is the first node (`prev == NULL`), update the `head` to the next node.
     - If the node is not the first node, update the `prev` node's `next` pointer to skip the node to be deleted.
     - If the node has a next node, update the `next` node's `prev` pointer to skip the node to be deleted.
   - Free the memory allocated for the node.
3. **Display List**:
   - Traverse the list from the head and print each node's data.
4. **Main Program**:
   - Initialize a doubly linked list with some nodes.
   - Call the delete function to remove a node with a specific value.
   - Display the list before and after deletion.

CODE:

```
// Write an algorithm and program to delete an item from double linked list. 7
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct Node {
    int data;
```

```
    struct Node *prev, *next;
};

void deleteNode(struct Node **head, int key) {
    struct Node *temp = *head;
    while(temp && temp->data != key)
        temp = temp->next;
    if(!temp) return;

    if(temp->prev)
        temp->prev->next = temp->next;
    else
        *head = temp->next;
    if(temp->next)
        temp->next->prev = temp->prev;

    free(temp);
}

void display(struct Node *node) {
    while(node) {
        printf("%d ", node->data);
        node = node->next;
    }
}

void main() {
    struct Node *head = NULL, *second = NULL, *third = NULL;
    clrscr();

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1; head->next = second; head->prev = NULL;
    second->data = 2; second->next = third; second->prev = head;
```

```
    third->data = 3; third->next = NULL; third->prev = second;

    printf("Original list: ");
    display(head);

    deleteNode(&head, 2);

    printf("\nList after deletion: ");
    display(head);

    getch();
}
```

OUTPUT :

```
Original list: 1 2 3
List after deletion: 1 3
```

PRACTICAL 8

Write an algorithm and program to Implementation of stack using array.

ALGORITHM:

1. **Initialize Stack**:
   - Declare an array `stack[MAX]` to hold the stack elements.
   - Set the variable `top` to `-1`, indicating the stack is initially empty.
2. **Push Operation**:
   - Check if the stack is full by comparing `top` with `MAX - 1`. If full, print "Stack Overflow".
   - Otherwise, increment `top` and assign the value to `stack[top]`.
3. **Pop Operation**:
   - Check if the stack is empty by checking if `top` is `-1`. If empty, print "Stack Underflow".
   - Otherwise, return the value at `stack[top]` and decrement `top`.
4. **Display Operation**:
   - If the stack is empty (`top == -1`), print "Stack is empty".
   - Otherwise, traverse from `top` to `0` and print each element.
5. **Menu**:
   - Use a loop to continuously prompt the user for a choice until they choose to exit. Choices include:
     - Push: Adds an item to the stack.
     - Pop: Removes the top item from the stack.
     - Display: Shows all items in the stack.
     - Exit: Terminates the program.

```
// Write an algorithm and program to Implementation of stack using
array. 8


#include <stdio.h>
```

```
#include <conio.h>
#define MAX 5

int stack[MAX], top = -1;

void push(int val) {
    if(top == MAX - 1)
        printf("Stack Overflow\n");
    else
        stack[++top] = val;
}

int pop() {
    if(top == -1) {
        printf("Stack Underflow\n");
        return -1;
    } else
        return stack[top--];
}

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

```
void main() {
    int choice, val;
    clrscr();

    do {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter value to push: ");
                    scanf("%d", &val);
                    push(val);
                    break;
            case 2: val = pop();
                    if(val != -1) printf("Popped value: %d\n", val);
                    break;
            case 3: display();
                    break;
        }
    } while(choice != 4);

    getch();
}
```

OUTPUT :

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 20

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack elements: 20 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Popped value: 20

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack elements: 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
```

PRACTICAL 9

Write an algorithm and program to Implementation of queue using array.

ALGORITHM:

1. **Define the Queue Structure**:
   - Use an array `queue[MAX]` to store the queue elements.
   - Use two variables `front` and `rear` to track the positions of the front and rear of the queue. Initialize both to `-1` to represent an empty queue.
2. **Enqueue Operation** (Insert an element into the queue):
   - Check if the queue is full (`rear == MAX - 1`). If full, print "Queue Overflow".
   - If the queue is not full, increment `rear` and add the element at `queue[rear]` .
   - If the queue is empty (`front == -1`), set `front = 0` .
3. **Dequeue Operation** (Remove an element from the queue):
   - Check if the queue is empty (`front == -1` or `front > rear`). If empty, print "Queue Underflow".
   - If the queue is not empty, return and remove the element at `queue[front]` and increment `front` .
4. **Display Operation**:
   - If the queue is empty, print "Queue is empty".
   - Otherwise, print the elements from `queue[front]` to `queue[rear]` .
5. **Main Program**:
   - Use a loop to display a menu and allow the user to perform enqueue, dequeue, or display operations.

CODE:

```
// Write an algorithm and program to Implementation of queue using
array. 9


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

```
#define MAX 5

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

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

int dequeue() {
    if(front == -1 || front > rear) {
        printf("Queue Underflow\n");
        return -1;
    } else
        return queue[front++];
}

void display() {
    int i;
    if(front == -1 || front > rear)
        printf("Queue is empty\n");
    else {
        printf("Queue elements: ");
        for(i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
```

```
}

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

    do {
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter value to enqueue: ");
                    scanf("%d", &val);
                    enqueue(val);
                    break;
            case 2: val = dequeue();
                    if(val != -1) printf("Dequeued value: %d\n", val);
                    break;
            case 3: display();
                    break;
        }
    } while(choice != 4);

    getch();
}
```

OUTPUT :

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 10

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 3
Queue elements: 10 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 2
Dequeued value: 10

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 3
Queue elements: 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 4
```

PRACTICAL 10

W.A.P. in C++ to exchange the value.

ALGORITHM:

1. **Define the Circular Queue Structure**:
   - Use an array `queue[MAX]` to store the queue elements.
   - Initialize two pointers: `front` and `rear` to `-1` to indicate an empty queue.
2. **Enqueue Operation** (Insert an element into the queue):
   - Check if the queue is full. This happens when `(rear + 1) % MAX == front`. If full, print "Queue Overflow".
   - If the queue is not full:
     - If `front == -1`, set `front` to `0` to start inserting.
     - Increment `rear` using `(rear + 1) % MAX` (this ensures circular behavior).
     - Add the element at `queue[rear]`.
3. **Dequeue Operation** (Remove an element from the queue):
   - Check if the queue is empty (`front == -1`). If empty, print "Queue Underflow".
   - If the queue is not empty, remove the element at `queue[front]`.
   - If `front == rear`, the queue becomes empty, so set both `front` and `rear` to `-1`.
   - Otherwise, increment `front` using `(front + 1) % MAX`.
4. **Display Operation**:
   - If the queue is empty, print "Queue is empty".
   - Otherwise, print the elements from `front` to `rear`, considering circular behavior.
5. **Main Program**:
   - Display a menu to allow the user to perform enqueue, dequeue, or display operations.

CODE:

```
// Write an algorithm and program to Implementation of circular queue
using array. 10


#include <stdio.h>
#include <conio.h>
#define MAX 5

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

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

int dequeue() {
    if(front == -1) {
        printf("Queue Underflow\n");
        return -1;
    } else {
        int val = queue[front];
        if(front == rear)
            front = rear = -1;
        else
            front = (front + 1) % MAX;
        return val;
    }
}
```

```
void display() {
    int i;
    if(front == -1)
        printf("Queue is empty\n");
    else {
        printf("Queue elements: ");
        for(i = front; i != rear; i = (i + 1) % MAX)
            printf("%d ", queue[i]);
        printf("%d\n", queue[rear]);
    }
}

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

    do {
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter value to enqueue: ");
                    scanf("%d", &val);
                    enqueue(val);
                    break;
            case 2: val = dequeue();
                    if(val != -1) printf("Dequeued value: %d\n", val);
                    break;
            case 3: display();
                    break;
```

```
        }
    } while(choice != 4);

    getch();
}
```

OUTPUT:

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 10

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 3
Queue elements: 10 20 30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 2
Dequeued value: 10

1. Enqueue
2. Dequeue
```

```
3. Display
4. Exit
Enter choice: 3
Queue elements: 20 30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 40

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 3
Queue elements: 20 30 40

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 4
```

PRACTICAL 11

Write an algorithm and program to Implementation of binary search tree using array.

ALGORITHM:

1. **Initialize Binary Search Tree (BST)**:
   - Create an array `bst[MAX]` to represent the tree.
   - Initialize all elements in the array as `-1` to signify empty positions.
2. **Insert Operation**:
   - Start at the root (`i = 0`).
   - If the current position (`bst[i]`) is `-1` (empty), insert the value at that position and return.
   - If the value to insert is less than the current node (`bst[i]`), move to the left child (`i = 2 * i + 1`).
   - If the value to insert is greater than the current node, move to the right child (`i = 2 * i + 2`).
   - If the array is full and no empty position is found, print "Tree is full."
3. **In-Order Traversal**:
   - Perform an in-order traversal of the tree, visiting the left child first, then the current node, and then the right child.
   - Print the values of the nodes during the traversal.
4. **Main Program**:
   - Display a menu for the user to insert values, perform in-order traversal, or exit.

CODE:

```
// Write an algorithm and program to Implementation of binary search tree using array. 11

#include <stdio.h>

#include <conio.h>
```

```
#define MAX 15

int bst[MAX];

// Function to initialize the binary search tree array
void initTree() {
    int i;
    for(i = 0; i < MAX; i++)
        bst[i] = -1;  // Initialize all elements as empty
}

// Function to insert a value into the binary search tree
void insert(int val) {
    int i = 0;
    while(i < MAX) {
        if(bst[i] == -1) {  // Insert at the first empty position
found
            bst[i] = val;
            return;
        } else if(val < bst[i])  // Move to the left child
            i = 2 * i + 1;
        else  // Move to the right child
            i = 2 * i + 2;
    }
    printf("Tree is full\n");
}

// Function for in-order traversal of the binary search tree
void inorder(int i) {
    if(i >= MAX || bst[i] == -1)
        return;
    inorder(2 * i + 1);      // Visit left child
```

```
    printf("%d ", bst[i]);   // Visit node
    inorder(2 * i + 2);      // Visit right child
}

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

    initTree();

    do {
        printf("1. Insert\n2. In-order Traversal\n3. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &val);
                insert(val);
                break;
            case 2:
                printf("In-order traversal: ");
                inorder(0);
                printf("\n");
                break;
        }
    } while(choice != 3);

    getch();  // Wait for keypress (Turbo C specific)
}
```