MCA – DATA STRUCTUR – RECORD BOOK

Part - A

Part A - Question 1: Program to Insert and Delete an Element at a Desired Position in an Array

Code :

```
#include <stdio.h>


#define MAX 100


void insertElement(int arr[], int *size, int pos, int
value) {
    if (*size >= MAX) {
        printf("Array is full!\n");
        return;
    }
    if (pos < 1 || pos > *size + 1) {
        printf("Invalid position!\n");
        return;
    }
    for (int i = *size; i >= pos; i--)
        arr[i] = arr[i - 1];
```

```
    arr[pos - 1] = value;
    (*size)++;
}

void deleteElement(int arr[], int *size, int pos) {
    if (*size <= 0) {
        printf("Array is empty!\n");
        return;
    }
    if (pos < 1 || pos > *size) {
        printf("Invalid position!\n");
        return;
    }
    for (int i = pos - 1; i < *size - 1; i++)
        arr[i] = arr[i + 1];

    (*size)--;
}

void displayArray(int arr[], int size) {
    printf("Array: ");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
```

```
    int arr[MAX], size, choice, pos, value;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);

    printf("Enter %d elements: ", size);
    for (int i = 0; i < size; i++)
        scanf("%d", &arr[i]);

    while (1) {
        printf("\nMenu:\n1. Insert Element\n2. Delete
Element\n3. Display Array\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter position and value to insert:
");
                scanf("%d %d", &pos, &value);
                insertElement(arr, &size, pos, value);
                break;
            case 2:
                printf("Enter position to delete: ");
                scanf("%d", &pos);
                deleteElement(arr, &size, pos);
                break;
            case 3:
                displayArray(arr, size);
```

```
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Enter the number of elements in the array: 5
Enter 5 elements: 10 20 30 40 50

Menu:
1. Insert Element
2. Delete Element
3. Display Array
4. Exit
Enter choice: 3
Array: 10 20 30 40 50

Enter choice: 1
Enter position and value to insert: 3 25
Array: 10 20 25 30 40 50

Enter choice: 2
Enter position to delete: 4
```

```
Array: 10 20 25 40 50

Enter choice: 4
```

Part A - Question 2: Program for 2D Array Manipulation and Matrix Multiplication

Code :

```
#include <stdio.h>

#define MAX 10

void inputMatrix(int mat[MAX][MAX], int row, int col) {
    printf("Enter elements of %dx%d matrix:\n", row, col);
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            scanf("%d", &mat[i][j]);
}

void displayMatrix(int mat[MAX][MAX], int row, int col) {
    printf("Matrix:\n");
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}

void multiplyMatrices(int mat1[MAX][MAX], int
mat2[MAX][MAX], int res[MAX][MAX], int r1, int c1, int c2)
{
    for (int i = 0; i < r1; i++)
```

```
        for (int j = 0; j < c2; j++) {
            res[i][j] = 0;
            for (int k = 0; k < c1; k++)
                res[i][j] += mat1[i][k] * mat2[k][j];
        }
}

int main() {
    int mat1[MAX][MAX], mat2[MAX][MAX], result[MAX][MAX];
    int r1, c1, r2, c2;

    printf("Enter rows and columns of first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and columns of second matrix: ");
    scanf("%d %d", &r2, &c2);

    if (c1 != r2) {
        printf("Matrix multiplication not possible.\n");
        return 1;
    }

    inputMatrix(mat1, r1, c1);
    inputMatrix(mat2, r2, c2);

    printf("First Matrix:\n");
    displayMatrix(mat1, r1, c1);
    printf("Second Matrix:\n");
```

```
    displayMatrix(mat2, r2, c2);

    multiplyMatrices(mat1, mat2, result, r1, c1, c2);

    printf("Resultant Matrix:\n");
    displayMatrix(result, r1, c2);

    return 0;
}
```

Output :

```
Enter rows and columns of first matrix: 2 3
Enter rows and columns of second matrix: 3 2

Enter elements of 2x3 matrix:
1 2 3
4 5 6

Enter elements of 3x2 matrix:
7 8
9 10
11 12

First Matrix:
1 2 3
4 5 6
```

```
Second Matrix:
7 8
9 10
11 12

Resultant Matrix:
58 64
139 154
```

Part A - Question 3: Program for String Operations (Compare, Concatenate, String Length) with Different Parameter Passing Techniques

Code :

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

#define MAX 100

int stringLength(char *str) {
    int length = 0;
    while (str[length] != '\0')
        length++;
    return length;
}

int stringCompare(char *str1, char *str2) {
    while (*str1 && *str2 && *str1 == *str2) {
        str1++;
        str2++;
    }
    return *str1 - *str2;
}

void stringConcatenate(char *str1, char *str2) {
    while (*str1)
        str1++;
```

```
    while (*str2)
        *str1++ = *str2++;
    *str1 = '\0';
}

int main() {
    char str1[MAX], str2[MAX];
    int choice;

    while (1) {
        printf("\nMenu:\n1. String Length\n2. String
Compare\n3. String Concatenate\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);
        getchar();  // Consume newline

        switch (choice) {
            case 1:
                printf("Enter a string: ");
                fgets(str1, MAX, stdin);
                str1[strcspn(str1, "\n")] = 0; // Remove
newline
                printf("String Length: %d\n",
stringLength(str1));
                break;

            case 2:
                printf("Enter first string: ");
                fgets(str1, MAX, stdin);
```

```
                str1[strcspn(str1, "\n")] = 0;
                printf("Enter second string: ");
                fgets(str2, MAX, stdin);
                str2[strcspn(str2, "\n")] = 0;
                printf("String Comparison Result: %d\n",
stringCompare(str1, str2));
                break;

            case 3:
                printf("Enter first string: ");
                fgets(str1, MAX, stdin);
                str1[strcspn(str1, "\n")] = 0;
                printf("Enter second string: ");
                fgets(str2, MAX, stdin);
                str2[strcspn(str2, "\n")] = 0;
                stringConcatenate(str1, str2);
                printf("Concatenated String: %s\n", str1);
                break;

            case 4:
                return 0;

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

Output :

```
Menu:
1. String Length
2. String Compare
3. String Concatenate
4. Exit
Enter choice: 1
Enter a string: Hello
String Length: 5

Enter choice: 2
Enter first string: Apple
Enter second string: Banana
String Comparison Result: -1

Enter choice: 3
Enter first string: Hello
Enter second string: World
Concatenated String: HelloWorld

Enter choice: 4
```

Part A - Question 4: Program to Swap Two Variables Using Call by Value and Call by Reference

Code :

```
#include <stdio.h>

void swapByValue(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    printf("After swap (Call by Value): a = %d, b = %d\n", a, b);
}

void swapByReference(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int a, b, choice;

    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);

    while (1) {
```

```
        printf("\nMenu:\n1. Swap using Call by Value\n2.
Swap using Call by Reference\n3. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                swapByValue(a, b);
                printf("Original values after function
call: a = %d, b = %d\n", a, b);
                break;
            case 2:
                swapByReference(&a, &b);
                printf("After swap (Call by Reference): a =
%d, b = %d\n", a, b);
                break;
            case 3:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Enter two numbers: 5 10

Menu:
1. Swap using Call by Value
```

```
2. Swap using Call by Reference
3. Exit
Enter choice: 1
After swap (Call by Value): a = 10, b = 5
Original values after function call: a = 5, b = 10

Enter choice: 2
After swap (Call by Reference): a = 10, b = 5

Enter choice: 3
```

Part A - Question 5: Program to Implement Recursive Function for Binary to Decimal Conversion

Code :

```
#include <stdio.h>

int binaryToDecimal(int binary) {
    if (binary == 0)
        return 0;
    return (binary % 10) + 2 * binaryToDecimal(binary / 10);
}

int main() {
    int binary;

    printf("Enter a binary number: ");
    scanf("%d", &binary);

    printf("Decimal equivalent: %d\n", binaryToDecimal(binary));

    return 0;
}
```

Output :

```
Enter a binary number: 1011
```

```
Decimal equivalent: 11

Enter a binary number: 11001
Decimal equivalent: 25
```

Part A - Question 6: Program to Implement Queue Operations (Insert, Delete, Display)

Code :

```
#include <stdio.h>

#define MAX 5

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

void enqueue(int value) {
    if (rear == MAX - 1) {
        printf("Queue is full!\n");
        return;
    }
    if (front == -1)
        front = 0;
    queue[++rear] = value;
    printf("%d inserted into the queue.\n", value);
}

void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty!\n");
        return;
    }
    printf("%d removed from the queue.\n", queue[front++]);
    if (front > rear)
```

```
        front = rear = -1;  // Reset queue if empty
}

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

int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n1. Enqueue\n2. Dequeue\n3. Display
Queue\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

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

```
            case 2:
                dequeue();
                break;
            case 3:
                displayQueue();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 1
Enter value to insert: 10
10 inserted into the queue.

Enter choice: 1
Enter value to insert: 20
20 inserted into the queue.
```

```
Enter choice: 3
Queue: 10 20

Enter choice: 2
10 removed from the queue.

Enter choice: 3
Queue: 20

Enter choice: 4
```

Part A - Question 7: Program to Sort a List of N Elements Using Quick Sort

Code :

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
```

```
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void displayArray(int arr[], int size) {
    printf("Sorted Array: ");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[100], n;

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

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

    quickSort(arr, 0, n - 1);
    displayArray(arr, n);

    return 0;
```

```
}
```

Output :

```
Enter number of elements: 5

Enter 5 elements: 34 7 23 32 5

Sorted Array: 5 7 23 32 34
```

Part A - Question 8: Program to Implement Search Techniques (Linear Search & Binary Search)

Code :

```
#include <stdio.h>

void linearSearch(int arr[], int size, int key) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            printf("Element %d found at position %d (Linear Search)\n", key, i + 1);
            return;
        }
    }
    printf("Element %d not found (Linear Search)\n", key);
}

int binarySearch(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
```

```
    return -1;
}

void bubbleSort(int arr[], int n) {  // Helper function to
sort for Binary Search
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}

int main() {
    int arr[100], size, key, choice;

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

    printf("Enter %d elements: ", size);
    for (int i = 0; i < size; i++)
        scanf("%d", &arr[i]);

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

    while (1) {
```

```
        printf("\nMenu:\n1. Linear Search\n2. Binary Search
(Sorted Required)\n3. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                linearSearch(arr, size, key);
                break;
            case 2:
                bubbleSort(arr, size);  // Sorting before
binary search
                int index = binarySearch(arr, 0, size - 1,
key);
                if (index != -1)
                    printf("Element %d found at position %d
(Binary Search)\n", key, index + 1);
                else
                    printf("Element %d not found (Binary
Search)\n", key);
                break;
            case 3:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Enter number of elements: 5
Enter 5 elements: 34 7 23 32 5
Enter element to search: 23

Menu:
1. Linear Search
2. Binary Search (Sorted Required)
3. Exit
Enter choice: 1
Element 23 found at position 3 (Linear Search)

Enter choice: 2
Element 23 found at position 3 (Binary Search)

Enter choice: 3
```

Part - B

Part B - Question 1: Program to Simulate Stack Operations (Push, Pop, Display)

Code :

```
#include <stdio.h>

#define MAX 5

int stack[MAX], top = -1;

void push(int value) {
    if (top == MAX - 1) {
        printf("Stack Overflow!\n");
        return;
    }
    stack[++top] = value;
    printf("%d pushed onto the stack.\n", value);
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow!\n");
        return;
    }
    printf("%d popped from the stack.\n", stack[top--]);
```

```
}

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

int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n1. Push\n2. Pop\n3. Display
Stack\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
```

```
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Menu:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter choice: 1
Enter value to push: 10
10 pushed onto the stack.

Enter choice: 1
Enter value to push: 20
20 pushed onto the stack.
```

```
Enter choice: 3
Stack: 20 10

Enter choice: 2
20 popped from the stack.

Enter choice: 3
Stack: 10

Enter choice: 4
```

Part B - Question 2: Program to Convert Infix Expression to Postfix Expression

Code :

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

#define MAX 100

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

void push(char ch) {
    if (top == MAX - 1) {
        printf("Stack Overflow!\n");
        return;
    }
    stack[++top] = ch;
}

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

int precedence(char ch) {
    if (ch == '+' || ch == '-') return 1;
```

```
    if (ch == '*' || ch == '/') return 2;
    if (ch == '^') return 3;
    return 0;
}

void infixToPostfix(char infix[]) {
    char postfix[MAX];
    int i = 0, j = 0;

    while (infix[i] != '\0') {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(infix[i]);
        } else if (infix[i] == ')') {
            while (top != -1 && stack[top] != '('
                postfix[j++] = pop();
            pop();  // Remove '('
        } else {
            while (top != -1 && precedence(stack[top]) >=
precedence(infix[i]))
                postfix[j++] = pop();
            push(infix[i]);
        }
        i++;
    }

    while (top != -1)
```

```
            postfix[j++] = pop();

        postfix[j] = '\0';
        printf("Postfix Expression: %s\n", postfix);
    }

    int main() {
        char infix[MAX];

        printf("Enter infix expression: ");
        scanf("%s", infix);

        infixToPostfix(infix);

        return 0;
    }
```

Output :

```
Enter infix expression: A+B*C
Postfix Expression: ABC*+
Enter infix expression: (A+B)*C
Postfix Expression: AB+C*
```

Part B - Question 3: Program to Simulate Circular Queue Operations (Insert, Delete, Display)

Code :

```
#include <stdio.h>

#define MAX 5

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

void enqueue(int value) {
    if ((rear + 1) % MAX == front) {
        printf("Queue is full!\n");
        return;
    }
    if (front == -1)
        front = 0;
    rear = (rear + 1) % MAX;
    queue[rear] = value;
    printf("%d inserted into the queue.\n", value);
}

void dequeue() {
    if (front == -1) {
        printf("Queue is empty!\n");
        return;
    }
```

```
    printf("%d removed from the queue.\n", queue[front]);
    if (front == rear)
        front = rear = -1;
    else
        front = (front + 1) % MAX;
}

void displayQueue() {
    if (front == -1) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear)
            break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
```

```
        printf("\nMenu:\n1. Enqueue\n2. Dequeue\n3. Display
Queue\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                displayQueue();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Menu:
1. Enqueue
```

```
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 1
Enter value to insert: 10
10 inserted into the queue.

Enter choice: 1
Enter value to insert: 20
20 inserted into the queue.

Enter choice: 1
Enter value to insert: 30
30 inserted into the queue.

Enter choice: 3
Queue: 10 20 30

Enter choice: 2
10 removed from the queue.

Enter choice: 3
Queue: 20 30

Enter choice: 4
```

Part B - Question 4: Program to Simulate Linked List Operations (Insert at Beginning, Insert at End, Delete at Beginning, Delete at End, Display)

Code :

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

typedef struct Node {
    int data;
    struct Node *next;
} Node;

Node *head = NULL;

void insertAtBeginning(int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
    printf("%d inserted at the beginning.\n", value);
}

void insertAtEnd(int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
```

```
    if (head == NULL) {
        head = newNode;
    } else {
        Node *temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    printf("%d inserted at the end.\n", value);
}

void deleteAtBeginning() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    Node *temp = head;
    head = head->next;
    printf("%d deleted from the beginning.\n", temp->data);
    free(temp);
}

void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
```

```
    if (head->next == NULL) {
        printf("%d deleted from the end.\n", head->data);
        free(head);
        head = NULL;
        return;
    }
    Node *temp = head;
    while (temp->next->next != NULL)
        temp = temp->next;
    printf("%d deleted from the end.\n", temp->next->data);
    free(temp->next);
    temp->next = NULL;
}

void displayList() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    Node *temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n1. Insert at Beginning\n2. Insert at End\n3. Delete at Beginning\n4. Delete at End\n5. Display List\n6. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 3:
                deleteAtBeginning();
                break;
            case 4:
                deleteAtEnd();
                break;
            case 5:
```

```
                displayList();
                break;
            case 6:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Delete at Beginning
4. Delete at End
5. Display List
6. Exit
Enter choice: 1
Enter value to insert: 10
10 inserted at the beginning.

Enter choice: 1
Enter value to insert: 20
20 inserted at the beginning.

Enter choice: 2
```

```
Enter value to insert: 30
30 inserted at the end.

Enter choice: 5
Linked List: 20 -> 10 -> 30 -> NULL

Enter choice: 3
20 deleted from the beginning.

Enter choice: 4
30 deleted from the end.

Enter choice: 5
Linked List: 10 -> NULL

Enter choice: 6
```

Part B - Question 5: Program to Sort a List of N Elements Using Insertion Sort

Code :

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void displayArray(int arr[], int n) {
    printf("Sorted Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[100], n;
```

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

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

    insertionSort(arr, n);
    displayArray(arr, n);

    return 0;
}
```

Output :

```
Enter number of elements: 5
Enter 5 elements: 34 7 23 32 5
Sorted Array: 5 7 23 32 34
```

Part B - Question 6: Program to Create a Binary Search Tree and Implement Tree Traversal (Inorder, Preorder, Postorder)

Code :

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

typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node* insert(Node* root, int value) {
    if (root == NULL)
        return createNode(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
```

```
    return root;
}

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

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

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

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

    while (1) {
        printf("\nMenu:\n1. Insert\n2. Inorder
Traversal\n3. Preorder Traversal\n4. Postorder
Traversal\n5. Exit\nEnter 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("Preorder Traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
```

```
                printf("Postorder Traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Menu:
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter choice: 1
Enter value to insert: 50

Enter choice: 1
Enter value to insert: 30

Enter choice: 1
Enter value to insert: 70
```

```
Enter choice: 2
Inorder Traversal: 30 50 70

Enter choice: 3
Preorder Traversal: 50 30 70

Enter choice: 4
Postorder Traversal: 30 70 50

Enter choice: 5
```

Part B - Question 7: Program to Implement B-Trees and Its Operations

Code :

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

#define MAX 3  // Maximum keys in a node (Order - 1)

typedef struct BTreeNode {
    int keys[MAX];
    struct BTreeNode *children[MAX + 1];
    int count;
    int leaf;
} BTreeNode;

BTreeNode* createNode(int leaf) {
    BTreeNode* node =
(BTreeNode*)malloc(sizeof(BTreeNode));
    node->count = 0;
    node->leaf = leaf;
    for (int i = 0; i <= MAX; i++)
        node->children[i] = NULL;
    return node;
}

void traverse(BTreeNode* root) {
    if (root) {
```

```
        int i;
        for (i = 0; i < root->count; i++) {
            if (!root->leaf)
                traverse(root->children[i]);
            printf("%d ", root->keys[i]);
        }
        if (!root->leaf)
            traverse(root->children[i]);
    }
}

BTreeNode* insertNonFull(BTreeNode* node, int key);
BTreeNode* splitChild(BTreeNode* parent, int i);

BTreeNode* insert(BTreeNode* root, int key) {
    if (!root) {
        root = createNode(1);
        root->keys[0] = key;
        root->count = 1;
        return root;
    }

    if (root->count == MAX) {
        BTreeNode* newRoot = createNode(0);
        newRoot->children[0] = root;
        newRoot = splitChild(newRoot, 0);
        newRoot = insertNonFull(newRoot, key);
```

```
        return newRoot;
    }

    return insertNonFull(root, key);
}

BTreeNode* insertNonFull(BTreeNode* node, int key) {
    int i = node->count - 1;

    if (node->leaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->count++;
    } else {
        while (i >= 0 && key < node->keys[i])
            i--;

        if (node->children[i + 1]->count == MAX) {
            node = splitChild(node, i + 1);
            if (key > node->keys[i + 1])
                i++;
        }
        node->children[i + 1] = insertNonFull(node->children[i + 1], key);
    }
```

```
    return node;
}

BTreeNode* splitChild(BTreeNode* parent, int i) {
    BTreeNode* child = parent->children[i];
    BTreeNode* newChild = createNode(child->leaf);

    newChild->keys[0] = child->keys[1];
    if (!child->leaf)
        newChild->children[0] = child->children[1];

    for (int j = parent->count; j > i; j--) {
        parent->keys[j] = parent->keys[j - 1];
        parent->children[j + 1] = parent->children[j];
    }

    parent->keys[i] = child->keys[0];
    parent->children[i + 1] = newChild;
    parent->count++;

    child->count = 1;

    return parent;
}

int main() {
```

```
    BTreeNode* root = NULL;
    int choice, key;

    while (1) {
        printf("\nMenu:\n1. Insert\n2. Display (Inorder
Traversal)\n3. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                root = insert(root, key);
                break;
            case 2:
                printf("B-Tree Inorder Traversal: ");
                traverse(root);
                printf("\n");
                break;
            case 3:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output :

```
Menu:
1. Insert
2. Display (Inorder Traversal)
3. Exit
Enter choice: 1
Enter key to insert: 10

Enter choice: 1
Enter key to insert: 20

Enter choice: 1
Enter key to insert: 5

Enter choice: 1
Enter key to insert: 15

Enter choice: 1
Enter key to insert: 30

Enter choice: 2
B-Tree Inorder Traversal: 5 10 15 20 30

Enter choice: 3
```

Part B - Question 8: Program to Implement Graph Traversal Techniques (DFS and BFS)

Code :

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

#define MAX 10

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

void addEdge(int u, int v) {
    graph[u][v] = 1;
    graph[v][u] = 1; // For undirected graph
}

void DFS(int vertex) {
    printf("%d ", vertex);
    visited[vertex] = 1;

    for (int i = 0; i < n; i++)
        if (graph[vertex][i] && !visited[i])
            DFS(i);
}

void enqueue(int value) {
    if (rear == MAX - 1)
```

```
        return;
    if (front == -1)
        front = 0;
    queue[++rear] = value;
}

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

void BFS(int start) {
    for (int i = 0; i < n; i++)
        visited[i] = 0;

    enqueue(start);
    visited[start] = 1;

    while (front <= rear) {
        int vertex = dequeue();
        printf("%d ", vertex);

        for (int i = 0; i < n; i++) {
            if (graph[vertex][i] && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
```

```
            }
        }
    }
}

int main() {
    int edges, u, v, start, choice;

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

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

    for (int i = 0; i < edges; i++) {
        printf("Enter edge (u v): ");
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    while (1) {
        printf("\nMenu:\n1. DFS Traversal\n2. BFS
Traversal\n3. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                for (int i = 0; i < n; i++)
```

```
                visited[i] = 0;
            printf("Enter starting vertex for DFS: ");
            scanf("%d", &start);
            printf("DFS Traversal: ");
            DFS(start);
            printf("\n");
            break;
        case 2:
            printf("Enter starting vertex for BFS: ");
            scanf("%d", &start);
            printf("BFS Traversal: ");
            BFS(start);
            printf("\n");
            break;
        case 3:
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

Output :

```
Enter number of vertices: 5
Enter number of edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
```

```
Enter edge (u v): 1 3
Enter edge (u v): 2 4

Menu:
1. DFS Traversal
2. BFS Traversal
3. Exit
Enter choice: 1
Enter starting vertex for DFS: 0
DFS Traversal: 0 1 3 2 4

Enter choice: 2
Enter starting vertex for BFS: 0
BFS Traversal: 0 1 2 3 4

Enter choice: 3
```
