

School of Engineering & Technology
Department of Computer Science & Technology



Data Structures Lab
(CSP 242)

Lab File
(2023-2024)

for

B. Tech. (CSE)
2nd Semester

Submitted To:

Dr. Pushpendra K Rajput
Associate Professor,
Department of Computer Science & Engineering
School of Engineering & Technology
Sharda University

Submitted By:

Aditya Pandey
B. Tech. CSE [2nd Semester]
2301010055
Group 1

INDEX

Serial No.	Exp. No.	Description	Page No.
1. Dynamic Memory allocation and Recursion			
1.	1.1	Create an integer array of user defined size n1 with dynamic memory allocation. Store data after reading from keyboard. Expand the size of array with n2. Read new values (n2 values from keyboard). Print state of array with all (n1+n2) values.	4
2.	1.2	Write a recursive function for Tower of Hanoi Problem.	5
3.	1.3	Write a function to find the sum of all array elements using recursion.	6
4.	1.4	Write a recursive function to print the reverse of a string.	7
5.	1.5	Write a program to create a 2-D array using dynamic memory allocation. Also write the code to scan the input and display element of created array.	7
2. Arrays			
6.	2.1	Write a menu driven C program to implement array operations (Insertion, Deletion, and Searching) on sorted array. Create the array dynamically with initial size n. if the array is found full increase the size of array as double (2 times of existing size) and insert the element.	9
7.	2.2	Write a menu driven program to perform the following operations on matrix. a. Addition of two matrices. b. Subtraction of two matrices. c. Multiplication of two matrices. d. Transpose of a matrix.	12
8.	2.3	Write a program to store a 2-D matrix and find the sum of each row and column.	15
9	2.4	Write a function that calculates the sum of both the diagonals of a given matrix. Use call by reference to update the variables for storing results.	17
3. Linear Link List Data Structure and its Applications			
10	3.1	Implement single Linked List data structure using array. Create all necessary functions to perform operations like insert and delete in the beginning/end and n th position of the list, and display the items stored in the linked list.	18
11	3.2	Implement single Linked List data structure and its operations like insert and delete in the beginning/end and n th position of the list, and display the items stored in the linked list.	23
	3.3	Add two polynomials using Linked List.	30
12	3.4	Implement doubly Linked List data structure and its operations like insert and delete in the beginning/end and n th position of the list, and display the items stored in the linked list.	33
13	3.5	Implement circular Linked List data structure and its operations like insert and delete in the beginning/end and n th position of the list, and display the items stored in the linked list.	38
4. Stack Data Structure			
14	4.1	Using array and functions implement Stack and its operations like push, pop, peek.	42
15	4.2	Use the stack operations developed in Prob 1 and reverse a string using stack	45

16	4.3	Using array and functions implement two Stacks and its operations (push, pop, peek).	46
17	4.4	Implement stack operations using linear linked list.	49
5. Queue Data Structure			
18	5.1	Using circular array and functions implement Queue data structure and its operations like insert, delete.	51
19	5.2	Implement Queue data structure using linked list and its operations (Enqueue, Dequeue, Display).	53
20	5.3	Check whether the string is palindrome or not using Stack and Queue.	55
21	5.4	Implement Double Ended Queue data structure using linked list. a. Input Restricted b. Output Restricted	56
22	5.5	Implement Priority Queue using array where the minimum element is having highest priority	58
23	5.6	Implement Priority Queue using Linked list where the priority is associated with each element.	60
6. Trees			
24	6.1	Create a binary tree using an array/linked List. Write the functions to perform Preorder, Inorder, Postorder and Level-order Traversal of constructed tree.	62
25	6.2	Write a Menu driven program to perform the following operations on Binary Search Tree. a. Insert b. Search c. Delete d. Traversals i. Inorder ii. Preorder iii. Postorder	64
26	6.3	Write a program to perform the following operations on Binary Search Tree a. Find Minimum Element b. Find Maximum Element	68
27	6.4	Write the program that reads the random sequence of integers and prints the sorted form of given data (ascending Order) using Binary Search Tree.	69
7. Sorting algorithms			
28	7.1	Read the numbers from a text file sort them into an array using ' <i>Insertion Sort</i> ' algorithm and write back in another text file.	71
29	7.2	Read the numbers from a text file sort them into an array using ' <i>Bubble Sort</i> ' algorithm and write back in another text file.	72
30	7.3	Read the numbers from a text file and write a function to search an element using linear search.	73
31	7.4	Read the numbers from a text file sort them into an array using ' <i>Selection Sort</i> ' algorithm and write back in another text file.	75
32	7.5	Read the numbers from a text file where numbers are stored in random manner. Write a program to search an element in the data using Binary Search. (Note: we require sorted data for applying binary Search Algorithm.	76
8. Graph			
33.	8.1	Create a graph in the memory and write a choice based program to demonstrate following two graph traversal Algorithm a. DFS (Depth First Search) b. BFS (Breadth First Search)	

Experiment-01

Title: 1- Dynamic Memory allocation and Recursion

Problem – 1.1: Create an integer array of user defined size n1 with dynamic memory allocation. Store data after reading from keyboard. Expand the size of array with n2. Read new values (n2 values from keyboard). Print state of array with all (n1+n2) values.

Source Code:

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

void readValues(int *arr, int n) {
    printf("Enter %d values:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
}

void expandArray(int **arr, int n1, int n2) {
    int *newArr = (int *)realloc(*arr, (n1 + n2) * sizeof(int));
    if (newArr == NULL) {
        printf("Memory reallocation failed\n");
        return;
    }

    *arr = newArr;
    printf("Enter %d new values:\n", n2);
    for (int i = n1; i < n1 + n2; i++) {
        scanf("%d", &(*arr)[i]);
    }
}

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

int main() {
    int n1, n2;
    printf("Enter the initial size of the array: ");
    scanf("%d", &n1);

    int *arr = (int *)malloc(n1 * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    readValues(arr, n1);
```

```

printf("Enter the number of new elements to add: ");
scanf("%d", &n2);

expandArray(&arr, n1, n2);

printArray(arr, n1 + n2);

free(arr);
return 0;
}

```

Sample Output:

```

Enter the initial size of the array: 5
Enter 5 values:
99
77
55
33
11
Enter the number of new elements to add: 4
Enter 4 new values:
88
66
44
22
Array elements:
99 77 55 33 11 88 66 44 22

```

Problem – 1.2: Write a recursive function for Tower of Hanoi Problem

Source Code:

```

#include <stdio.h>

void TOH(int n , char from , char to , char help){
    if (n==0) return;
    TOH(n-1,from , help , to);
    printf("Move %d from %c -> %c\n",n,from , to);
    TOH(n-1 , help , to , from);
}

```

```

}

int main() {
    int N = 3;
    TOH(N, 'A', 'C', 'B');
    return 0;
}

```

Sample Output:

```

Move 1 from A -> C
Move 2 from A -> B
Move 1 from C -> B
Move 3 from A -> C
Move 1 from B -> A
Move 2 from B -> C
Move 1 from A -> C

```

Problem - 1.3: Write a function to find the sum of all array elements using recursion.

Source Code:

```

#include <stdio.h>

int sumArray(int arr[], int n) {
    if (n == 0) {
        return 0;
    }
    return arr[n - 1] + sumArray(arr, n - 1);
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    int sum = sumArray(arr, size);
    printf("Sum of array elements: %d\n", sum);

    return 0;
}

```

Sample Output:

```

Sum of array elements: 15

```

Problem - 1.4: Write a recursive function to print the reverse of a string.

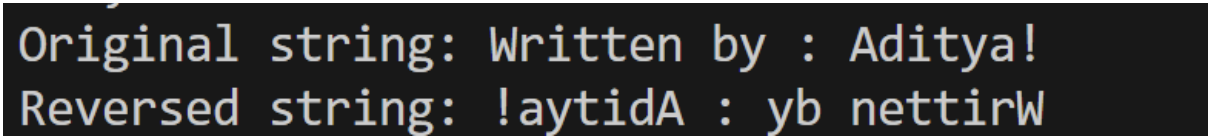
Source Code:

```
#include <stdio.h>

void reverseString(char str[]) {
    if (str[0] == '\0') {
        return;
    }
    reverseString(&str[1]);
    printf("%c", str[0]);
}

int main() {
    char str[] = "Written by : Aditya!";
    printf("Original string: %s\n", str);
    printf("Reversed string: ");
    reverseString(str);
    printf("\n");
    return 0;
}
```

Sample Output:



```
Original string: Written by : Aditya!
Reversed string: !aytidA : yb nettirW
```

Problem - 2.5: Write a program to create a 2-D array using dynamic memory allocation. Also write the code to scan the input and display element of created array.

Source Code:

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

int main() {
    int rows, cols;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);

    int **arr = (int **)malloc(rows * sizeof(int *));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

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

```

        arr[i] = (int *)malloc(cols * sizeof(int));
        if (arr[i] == NULL) {
            printf("Memory allocation failed\n");
            return 1;
        }
    }

    printf("Enter elements of the %d x %d array:\n", rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &arr[i][j]);
        }
    }

    printf("Elements of the %d x %d array:\n", rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }

    for (int i = 0; i < rows; i++) {
        free(arr[i]);
    }
    free(arr);

    return 0;
}

```

Sample Output:

```

Enter number of rows: 3
Enter number of columns: 4
Enter elements of the 3 x 4 array:
9 8 9 7 4 5 6 8 2 4 1 7
Elements of the 3 x 4 array:
9 8 9 7
4 5 6 8
2 4 1 7

```

Experiment-02

Title: 2- Array Operation on Sorted Array.

Objective: To apply the concept of insertion, deletion, and binary search on a sorted array.

Problem – 2.1: Write a menu driven C program to implement array operations (Insertion, Deletion, Searching) on sorted array. Create the array dynamically with initial size n. if the array is found full increase the size of array as double (2 times of existing size) and insert the element.

Source Code:

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

void insertElement(int **arr, int *size, int *capacity, int element);
void deleteElement(int **arr, int *size, int element);
int searchElement(int *arr, int size, int element);
void displayArray(int *arr, int size);

int main() {
    int *arr = NULL;
    int size = 0;
    int capacity = 0;
    int choice, element;

    do {
        printf("\n----- MENU ----- \n");
        printf("1. Insert Element\n");
        printf("2. Delete Element\n");
        printf("3. Search Element\n");
        printf("4. Display Array\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to insert: ");
                scanf("%d", &element);
                insertElement(&arr, &size, &capacity, element);
                break;
            case 2:
                printf("Enter element to delete: ");
                scanf("%d", &element);
                deleteElement(&arr, &size, element);
                break;
            case 3:
                printf("Enter element to search: ");
                scanf("%d", &element);
                if (searchElement(arr, size, element) != -1)
                    printf("Element %d found in the array.\n", element);
                else
                    printf("Element %d not found in the array.\n",
element);
                break;
            case 4:
                displayArray(arr, size);
                break;
            case 5:
```

```

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

    } while (choice != 5);

    free(arr);

    return 0;
}

void insertElement(int **arr, int *size, int *capacity, int element) {
    if (*size == *capacity) {
        *capacity = (*capacity == 0) ? 1 : *capacity * 2;
        *arr = (int *)realloc(*arr, *capacity * sizeof(int));
    }

    int i = *size - 1;
    while (i >= 0 && (*arr)[i] > element) {
        (*arr)[i + 1] = (*arr)[i];
        i--;
    }

    (*arr)[i + 1] = element;
    (*size)++;
    printf("Element %d inserted successfully.\n", element);
}

void deleteElement(int **arr, int *size, int element) {
    int pos = searchElement(*arr, *size, element);
    if (pos != -1) {
        for (int i = pos; i < *size - 1; i++) {
            (*arr)[i] = (*arr)[i + 1];
        }
        (*size)--;
        printf("Element %d deleted successfully.\n", element);
    } else {
        printf("Element %d not found in the array.\n", element);
    }
}

int searchElement(int *arr, int size, int element) {
    int low = 0, high = size - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == element)
            return mid;
        else if (arr[mid] < element)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

```

```

}

void displayArray(int *arr, int size) {
    if (size == 0) {
        printf("Array is empty.\n");
    } else {
        printf("Array elements: ");
        for (int i = 0; i < size; i++) {
            printf("%d ", arr[i]);
        }
        printf("\n");
    }
}
}

```

Sample Output:

```

----- MENU -----
1. Insert Element
2. Delete Element
3. Search Element
4. Display Array
5. Exit
Enter your choice: 1
Enter element to insert: 55
Element 55 inserted successfully.

----- MENU -----
1. Insert Element
2. Delete Element
3. Search Element
4. Display Array
5. Exit
Enter your choice: 1
Enter element to insert: 98
Element 98 inserted successfully.

----- MENU -----
1. Insert Element
2. Delete Element
3. Search Element
4. Display Array
5. Exit
Enter your choice: 4
Array elements: 55 98

----- MENU -----
1. Insert Element
2. Delete Element
3. Search Element
4. Display Array
5. Exit
Enter your choice: 3
Enter element to search: 98
Element 98 found in the array.

----- MENU -----
1. Insert Element
2. Delete Element
3. Search Element
4. Display Array
5. Exit
Enter your choice: 2
Enter element to delete: 55
Element 55 deleted successfully.

----- MENU -----
1. Insert Element
2. Delete Element
3. Search Element
4. Display Array
5. Exit
Enter your choice: 4
Array elements: 98

```

Problem - 2.2: Write a menu driven program to perform the following operations on matrix.

- a. Addition of two matrices.
- b. Subtraction of two matrices.
- c. Multiplication of two matrices.
- d. Transpose of a matrix.

Source Code:

```
#include <stdio.h>

void matrix_addition(int A[10][10], int B[10][10], int rows, int cols) {
    int result[10][10];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }
    printf("\nResult of A + B:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
}

void matrix_subtraction(int A[10][10], int B[10][10], int rows, int cols)
{
    int result[10][10];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }
    printf("\nResult of A - B:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
}

void matrix_multiplication(int A[10][10], int B[10][10], int rowsA, int
colsA, int colsB) {
    int result[10][10];
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            result[i][j] = 0;
            for (int k = 0; k < colsA; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

```

    }
    printf("\nResult of A * B:\n");
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
}

void transpose_matrix(int A[10][10], int rows, int cols) {
    int transposed[10][10];
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            transposed[i][j] = A[j][i];
        }
    }
    printf("\nTranspose of Matrix A:\n");
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            printf("%d ", transposed[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int choice, rowsA, colsA, rowsB, colsB;
    int A[10][10], B[10][10];

    // Input Matrix A
    printf("Enter number of rows and columns for Matrix A: ");
    scanf("%d %d", &rowsA, &colsA);
    printf("Enter elements of Matrix A:\n");
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsA; j++) {
            scanf("%d", &A[i][j]);
        }
    }

    // Input Matrix B
    printf("Enter number of rows and columns for Matrix B: ");
    scanf("%d %d", &rowsB, &colsB);
    printf("Enter elements of Matrix B:\n");
    for (int i = 0; i < rowsB; i++) {
        for (int j = 0; j < colsB; j++) {
            scanf("%d", &B[i][j]);
        }
    }

    while (1) {
        printf("\nMatrix Operations Menu:\n");
        printf("1. Addition of two matrices\n");
        printf("2. Subtraction of two matrices\n");
        printf("3. Multiplication of two matrices\n");
    }
}

```

```

printf("4. Transpose of Matrix A\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        if (rowsA == rowsB && colsA == colsB) {
            matrix_addition(A, B, rowsA, colsA);
        } else {
            printf("Matrices cannot be added. Dimensions must be
the same.\n");
        }
        break;

    case 2:
        if (rowsA == rowsB && colsA == colsB) {
            matrix_subtraction(A, B, rowsA, colsA);
        } else {
            printf("Matrices cannot be subtracted. Dimensions must
be the same.\n");
        }
        break;

    case 3:
        if (colsA == rowsB) {
            matrix_multiplication(A, B, rowsA, colsA, colsB);
        } else {
            printf("Matrices cannot be multiplied. Inner
dimensions must match.\n");
        }
        break;

    case 4:
        transpose_matrix(A, rowsA, colsA);
        break;

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

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

return 0;
}

```

Sample Output:

```
Enter number of rows and columns for Matrix A: 2 2
Enter elements of Matrix A:
9 7 5 3
Enter number of rows and columns for Matrix B: 2 2
Enter elements of Matrix B:
2 4 6 8
```

```
Matrix Operations Menu:
1. Addition of two matrices
2. Subtraction of two matrices
3. Multiplication of two matrices
4. Transpose of Matrix A
5. Exit
Enter your choice: 1
```

```
Result of A + B:
11 11
11 11
```

```
Matrix Operations Menu:
1. Addition of two matrices
2. Subtraction of two matrices
3. Multiplication of two matrices
4. Transpose of Matrix A
5. Exit
Enter your choice: 2
```

```
Result of A - B:
7 3
-1 -5
```

```
Matrix Operations Menu:
1. Addition of two matrices
2. Subtraction of two matrices
3. Multiplication of two matrices
4. Transpose of Matrix A
5. Exit
```

Enter your choice: 3

```
Result of A * B:
60 92
28 44
```

```
Matrix Operations Menu:
1. Addition of two matrices
2. Subtraction of two matrices
3. Multiplication of two matrices
4. Transpose of Matrix A
5. Exit
```

Enter your choice: 4

```
Transpose of Matrix A:
9 5
7 3
```

Problem – 2.3: Write a program to store a 2-D matrix and find the sum of each row and column.

Source Code:

```
#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10

void inputMatrix(int matrix[MAX_ROWS][MAX_COLS], int rows, int cols) {
    printf("Enter elements of the matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
}

void printMatrix(int matrix[MAX_ROWS][MAX_COLS], int rows, int cols) {
    printf("The matrix is:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

void calculateRowSums(int matrix[MAX_ROWS][MAX_COLS], int rows, int cols) {
    printf("\nRow sums:\n");
    for (int i = 0; i < rows; i++) {
        int rowSum = 0;
        for (int j = 0; j < cols; j++) {
```

```

        rowSum += matrix[i][j];
    }
    printf("Sum of Row %d: %d\n", i + 1, rowSum);
}
}

void calculateColumnSums(int matrix[MAX_ROWS][MAX_COLS], int rows, int
cols) {
    printf("\nColumn sums:\n");
    for (int j = 0; j < cols; j++) {
        int colSum = 0;
        for (int i = 0; i < rows; i++) {
            colSum += matrix[i][j];
        }
        printf("Sum of Column %d: %d\n", j + 1, colSum);
    }
}

int main() {
    int matrix[MAX_ROWS][MAX_COLS];
    int rows, cols;

    printf("Enter number of rows (max %d) and columns (max %d) for the
matrix: ", MAX_ROWS, MAX_COLS);
    scanf("%d %d", &rows, &cols);

    if (rows <= 0 || rows > MAX_ROWS || cols <= 0 || cols > MAX_COLS) {
        printf("Invalid input for rows or columns. Exiting...\n");
        return 1;
    }

    inputMatrix(matrix, rows, cols);

    printMatrix(matrix, rows, cols);

    calculateRowSums(matrix, rows, cols);

    calculateColumnSums(matrix, rows, cols);

    return 0;
}

```

Sample Output:


```

Enter number of rows (max 10) and columns (max 10) for the matrix: 2
2
Enter elements of the matrix:
1 2 3 4
The matrix is:
1 2
3 4

Row sums:
Sum of Row 1: 3
Sum of Row 2: 7

Column sums:
Sum of Column 1: 4
Sum of Column 2: 6

```

Problem – 2.4: Write a function that calculates the sum of both the diagonals of a given matrix. Use call by reference to update the variables for storing results.

Source Code:

```

#include <stdio.h>
#define MAX_SIZE 10

void sumDiagonals(int matrix[MAX_SIZE][MAX_SIZE], int size, int
*primaryDiagonalSum, int *secondaryDiagonalSum) {
    *primaryDiagonalSum = 0;
    *secondaryDiagonalSum = 0;

    for (int i = 0; i < size; i++) {
        *primaryDiagonalSum += matrix[i][i];
        *secondaryDiagonalSum += matrix[i][size - 1 - i];
    }
}

int main() {
    int matrix[MAX_SIZE][MAX_SIZE];
    int size;

    printf("Enter size of the square matrix (max %d): ", MAX_SIZE);
    scanf("%d", &size);

    printf("Enter elements of the matrix:\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    int primaryDiagonalSum, secondaryDiagonalSum;
    sumDiagonals(matrix, size, &primaryDiagonalSum,
&secondaryDiagonalSum);

```

```

printf("Sum of primary diagonal: %d\n", primaryDiagonalSum);
printf("Sum of secondary diagonal: %d\n", secondaryDiagonalSum);

return 0;
}

```

Sample Output:

```

Enter size of the square matrix (max 10): 3
Enter elements of the matrix:
9 4 6 2 1 3 7 8 3
Sum of primary diagonal: 13
Sum of secondary diagonal: 14

```

Experiment-03

Title: Linear Link List Data Structure and its Applications

Objective: To implement linear linked list data structure in C using structures, pointers, and dynamic memory allocation.

Problem - 3.1: Implement single Linked List data structure using array. Create all necessary functions to perform operations like insert and delete in the beginning/end and n^{th} position of the list, and display the items stored in the linked list.

Source Code:

```

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

#define MAX_SIZE 100

int DATA[MAX_SIZE]; // Array to store node data
int LINK[MAX_SIZE];  // Array to store next node indices
int start = -1;      // Index of the first node in the linked list
int Avail = 0;       // Index to manage available nodes in the array

void traversing() {
    int PTR = start;
    printf("Linked List: ");
    while (PTR != -1) {
        printf("%d -> ", DATA[PTR]);
        PTR = LINK[PTR];
    }
    printf("NULL\n");
}

void countLink() {
    int ptr = start;
    int COUNT = 0;
    while (ptr != -1) {
        COUNT++;
        ptr = LINK[ptr];
    }
}

```

```

    }
    printf("Number of Nodes: %d\n", COUNT);
}

int search(int ITEM) {
    int ptr = start;
    while (ptr != -1 && DATA[ptr] != ITEM) {
        ptr = LINK[ptr];
    }
    return (ptr != -1 && DATA[ptr] == ITEM) ? ptr : -1;
}

void insertSpecific(int Item, int Pos) {
    if (Avail == -1) {
        printf("Overflow: Linked list is full.\n");
        return;
    }

    int new_node = Avail;
    Avail = LINK[Avail]; // Update Avail to next available node

    if (Pos == 1) {
        LINK[new_node] = start;
        start = new_node;
    } else {
        int ptr = start;
        int count = 1;
        while (count < Pos - 1 && ptr != -1) {
            ptr = LINK[ptr];
            count++;
        }
        if (ptr == -1) {
            printf("Invalid position to insert.\n");
            return;
        }
        LINK[new_node] = LINK[ptr];
        LINK[ptr] = new_node;
    }
    DATA[new_node] = Item;
}

void insert_at_first(int item) {
    insertSpecific(item, 1);
}

void insert_at_End(int item) {
    int pos = 1;
    int ptr = start;
    while (ptr != -1) {
        ptr = LINK[ptr];
        pos++;
    }
    insertSpecific(item, pos);
}

```

```

void delete_Specific(int loc, int ploc) {
    if (start == -1) {
        printf("Underflow: Linked list is empty.\n");
        return;
    }
    if (ploc == -1) {
        start = LINK[start];
    } else {
        LINK[ploc] = LINK[loc];
    }
    LINK[loc] = Avail;
    Avail = loc;
}

void del_First() {
    delete_Specific(start, -1);
}

void del_Last() {
    if (start == -1) {
        printf("Underflow: Linked list is empty.\n");
        return;
    }
    int loc = start;
    int ploc = -1;
    while (LINK[loc] != -1) {
        ploc = loc;
        loc = LINK[loc];
    }
    if (ploc == -1) {
        start = LINK[start];
    } else {
        LINK[ploc] = -1;
    }
    LINK[loc] = Avail;
    Avail = loc;
}

void del_item(int Item) {
    int ploc = -1;
    int loc = start;
    while (loc != -1 && DATA[loc] != Item) {
        ploc = loc;
        loc = LINK[loc];
    }
    if (loc == -1) {
        printf("Item %d not found in the linked list.\n", Item);
        return;
    }
    if (ploc == -1) {
        start = LINK[loc];
    } else {
        LINK[ploc] = LINK[loc];
    }
}

```

```

    LINK[loc] = Avail;
    Avail = loc;
    printf("Item %d deleted from the linked list.\n", Item);
}

int main() {
    for (int i = 0; i < MAX_SIZE - 1; i++) {
        LINK[i] = i + 1; // Initialize all LINKs to point to the next
index
    }
    LINK[MAX_SIZE - 1] = -1; // Last LINK points to -1 indicating end of
list

    int choice, item, pos;
    while (1) {
        printf("\n\n1. Display\n2. Count Nodes\n3. Search\n4. Insert At
Beginning\n5. Insert At End\n6. Insert At Specific Position\n7. Delete the
First Node\n8. Delete the Last Node\n9. Delete Specific Node\n10. Delete
Node By its Data\n11. EXIT\n");
        printf("Enter Your Choice: ");
        scanf("%d", &choice);

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

            case 2:
                countLink();
                break;

            case 3:
                printf("Enter the value to be Searched: ");
                scanf("%d", &item);
                pos = search(item);
                if (pos != -1) {
                    printf("Item found at position %d.\n", pos + 1);
                } else {
                    printf("Item not found in the linked list.\n");
                }
                break;

            case 4:
                printf("Enter the value to Insert at Start: ");
                scanf("%d", &item);
                insert_at_first(item);
                break;

            case 5:
                printf("Enter the value to Insert at End: ");
                scanf("%d", &item);
                insert_at_End(item);
                break;

            case 6:

```

```

        printf("Enter the position of node to insert: ");
        scanf("%d", &pos);
        printf("Enter the Data to insert in node: ");
        scanf("%d", &item);
        insertSpecific(item, pos);
        break;

    case 7:
        del_First();
        break;

    case 8:
        del_Last();
        break;

    case 9:
        printf("Enter the position of node: ");
        scanf("%d", &pos);
        delete_Specific(pos - 1, -1); // Convert position to
zero-indexed
        break;

    case 10:
        printf("Enter the Node's Item: ");
        scanf("%d", &item);
        del_item(item);
        break;

    case 11:
        exit(0);

    default:
        printf("Invalid choice. Please enter a valid option.\n");
        break;
    }
}

return 0;
}

```

Sample Output:

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 7
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 1
Linked List: 55 -> NULL
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 10
Enter the Node's Item: 55
Item 55 deleted from the linked list.
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 1
Linked List: NULL
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 4
Enter the value to Insert at Start: 55
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 4
Enter the value to Insert at Start: 99
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 1
Linked List: 99 -> 55 -> NULL
```

```
1. Display
2. Count Nodes
3. Search
4. Insert At Beginning
5. Insert At End
6. Insert At Specific Position
7. Delete the First Node
8. Delete the Last Node
9. Delete Specific Node
10. Delete Node By its Data
11. EXIT
Enter Your Choice: 3
Enter the value to be Searched: 55
Item found at position 1.
```

Problem - 3.2: Implement single Linked List data structure and its operations like insert and delete in the beginning/end and n^{th} position of the list, and display the items stored in the linked list.

Source Code:

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

struct node{
    int Data;
    struct node*Link;
};

struct node *Create_Node(){
    return (struct node*)malloc(sizeof(struct node));
}

void traverse(struct node *start){

    struct node *ptr;
    ptr = start;
    while(ptr!=NULL){
        printf("%d -> ",ptr->Data);
        ptr = ptr -> Link ;
    }
    printf("NULL\n");
}

void countNodes(struct node *start) {
    int count = 0;
    struct node *ptr;
    ptr = start;
    while (ptr != NULL) {
        count++;
        ptr = ptr->Link;
    }
    printf("No of Nodes are : %d\n", count);
}

void Search(struct node *start, int Item){
    int count = 1;
    struct node *ptr;
    ptr = start;
    while (ptr != NULL && ptr->Data != Item){
        count++;
        ptr = ptr->Link;
    }

    if (ptr == NULL){
        printf("Element Not found");
    }
    else{
        printf("%d found at %dth node.\n", Item, count);
    }
}

void insert_begining(struct node **start , int item){
    struct node *n1 = Create_Node();
    if (n1 == NULL) {
```



```

        printf("Memory allocation failed.\n");
        exit(1);
    }

    n1->Data = item;
    n1->Link = *start;
    *start = n1;
}

void insert_at_End(struct node **start, int item){
    struct node *n1 = Create_Node();

    if (n1 == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    n1->Data = item;
    n1->Link = NULL;

    if(*start == NULL) *start = n1;
    else{
        struct node *ptr;
        ptr = *start;

        while(ptr->Link != NULL){
            ptr = ptr->Link;
        }
        ptr->Link = n1;
    }
}

void insertSpecific(struct node **start , int item , int pos){
    struct node *n1 = Create_Node();
    if (n1 == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    n1->Data = item;
    n1->Link = NULL;

    if(pos == 0){
        n1->Link = *start;
        *start = n1;
    }
    else{
        struct node *ptr;
        ptr = *start;
        int currentPos = 0;

        while(ptr!= NULL && currentPos != pos-2){
            ptr = ptr->Link;
            currentPos++;
        }
    }
}

```

```

        if (ptr == NULL) {
            printf("Invalid position for insertion.\n");
            free(n1); // Release the
allocated memory
            return;
        }
        n1->Link = ptr->Link;
        ptr->Link = n1;
    }
}

void del_begining(struct node **start){
    if(*start == NULL){
        printf("UnderFlow");
        return;
    }

    struct node *ptr;
    ptr = *start;
    *start = (*start)->Link;
    free(ptr);
}

void del_end(struct node **start){

    if(*start == NULL){
        printf("UnderFlow");
        return;
    }

    struct node *ptr = *start;
    struct node *pptr = NULL;

    while (ptr->Link != NULL){
        pptr = ptr;
        ptr = ptr->Link;
    }

    if(ptr == *start) *start = NULL;

    // Remove the last node from the list
    else pptr->Link = NULL;

    // Free the memory of deleted node.
    free(ptr);
}

void del_specific(struct node **start , int pos){
    if(*start == NULL){
        printf("UnderFlow");
        return;
    }
    int counter = 0;
    struct node *ptr = *start;
    struct node *prev = NULL;

```

```

while(ptr != NULL && counter != pos - 1){
    counter++;
    prev = ptr;
    ptr = ptr->Link;
}
if(ptr == NULL){
    printf("Invalid Position");
    free(ptr);
    return;
}

if(ptr == *start) *start = ptr->Link;
else prev->Link = ptr->Link;

free(ptr);
}

void del_item(struct node **start , int item){
    if(*start == NULL){
        printf("UnderFlow");
        return;
    }

    struct node *ptr = *start;
    struct node *prev = NULL;

    while(ptr != NULL && ptr->Data != item){
        prev = ptr;
        ptr = ptr->Link;
    }

    if(ptr == NULL){
        printf("Invalid item (%d not present in the List).\n",item);
        return;
    }

    if(prev == NULL) *start = ptr->Link;
    else prev->Link = ptr->Link;
    free(ptr);
}

int main(){
    struct node *n1, *start = NULL, *ptr;
    int choice, item, pos;
    while (1){
        printf("\n\n1.Display \n2.Count Nodes \n3.Search \n4.Insert At
Beginning \n5.Inser At Last \n6.Insert At Specific. \n7.Delete the First
Node \n8.Delete the End Node \n9.Delete Specific Node \n10.Delete Node By
its Data\n11.EXIT\n");
        printf("Enter Your Choice :");
        scanf("%d", &choice);
        switch (choice){
            case 1:

```

```

        traverse(start);
        printf("\n");
        break;

case 2:
    countNodes(start);
    printf("\n");
    break;

case 3:
    printf("Enetr the value to be Searched : ");
    scanf("%d", &item);
    Search(start, item);
    printf("\n");
    break;

case 4:
    printf("Enetr the value to Insert at Start : ");
    scanf("%d", &item);
    insert_begining(&start, item);
    printf("\n");
    break;

case 5:
    printf("Enetr the value to Insert at End : ");
    scanf("%d", &item);
    insert_at_End(&start, item);
    printf("\n");
    break;

case 6:
    printf("Enetr the position of node to insert : ");
    scanf("%d", &pos);
    printf("\n");
    printf("Enetr the Data to insert in node : ");
    scanf("%d", &item);
    insertSpecific(&start, item, pos);
    printf("\n");
    break;

case 7:
    del_begining(&start);
    printf("\n");
    break;

case 8:
    del_end(&start);
    printf("\n");
    break;

case 9:
    printf("Enetr the position of node : ");
    scanf("%d", &pos);
    del_specific(&start, pos);
    printf("\n");

```

```

        break;

    case 10:
        printf("Enetr the Node's Item : ");
        scanf("%d", &item);
        del_item(&start, item);
        printf("\n");
        break;

    case 11:
        exit(0);
        break;

    default:
        printf("Invalid choice. Please enter a valid option.\n");
        break;
}

return 0;
}

```

Sample Output:

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :10
Enetr the Node's Item : 1

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :1
11 -> 90 -> NULL

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :8

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :1
11 -> NULL

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :2
No of Nodes are : 1

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :4
Enetr the value to Insert at Start : 11

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :5
Enetr the value to Insert at End : 90

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :1
11 -> 90 -> NULL

```

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :6
Enetr the position of node to insert : 2

```

Enetr the Data to insert in node : 1

```

1.Display
2.Count Nodes
3.Search
4.Insert At Begining
5.Inser At Last
6.Insert At Specific.
7.Delete the First Node
8.Delete the End Node
9.Delete Specific Node
10.Delete Node By its Data
11.EXIT
Enter Your Choice :1
11 -> 1 -> 90 -> NULL

```

Problem - 3.3: Add two polynomials using Linked List.

Source Code:

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

struct Node {
    int coefficient;
    int power;
    struct Node* next;
};

struct Node* createNode(int coeff, int pow) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coeff;
    newNode->power = pow;
    newNode->next = NULL;
    return newNode;
}

void insertTerm(struct Node** poly, int coeff, int pow) {
    struct Node* newNode = createNode(coeff, pow);

    if (*poly == NULL) {
        *poly = newNode;
    } else {
        struct Node* temp = *poly;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void displayPolynomial(struct Node* poly) {
    struct Node* temp = poly;
    while (temp != NULL) {
        printf("%dx^d", temp->coefficient, temp->power);
        temp = temp->next;
        if (temp != NULL) {
            printf(" + ");
        }
    }
    printf("\n");
}

struct Node* addPolynomials(struct Node* poly1, struct Node* poly2) {
    struct Node* result = NULL;
    struct Node* temp1 = poly1;
    struct Node* temp2 = poly2;

    while (temp1 != NULL && temp2 != NULL) {
```

```

        if (temp1->power == temp2->power) {
            insertTerm(&result, temp1->coefficient + temp2->coefficient,
temp1->power);
            temp1 = temp1->next;
            temp2 = temp2->next;
        } else if (temp1->power > temp2->power) {
            insertTerm(&result, temp1->coefficient, temp1->power);
            temp1 = temp1->next;
        } else {
            insertTerm(&result, temp2->coefficient, temp2->power);
            temp2 = temp2->next;
        }
    }

    while (temp1 != NULL) {
        insertTerm(&result, temp1->coefficient, temp1->power);
        temp1 = temp1->next;
    }

    while (temp2 != NULL) {
        insertTerm(&result, temp2->coefficient, temp2->power);
        temp2 = temp2->next;
    }

    return result;
}

void cleanup(struct Node* poly) {
    struct Node* current = poly;
    struct Node* next;

    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
}

int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;

    int choice;
    do {
        printf("\nPolynomial Menu:\n");
        printf("1. Enter Polynomial 1\n");
        printf("2. Enter Polynomial 2\n");
        printf("3. Display Polynomial 1\n");
        printf("4. Display Polynomial 2\n");
        printf("5. Add Polynomials\n");
        printf("6. Exit\n");

        printf("Enter Your Choice: ");
        scanf("%d", &choice);
    }

```

```

switch (choice) {
    case 1: {
        int coeff, power;
        printf("Enter the coefficient and power for Polynomial 1
(Enter -1 -1 to stop):\n");
        while (1) {
            scanf("%d %d", &coeff, &power);
            if (coeff == -1 && power == -1) {
                break;
            }
            insertTerm(&poly1, coeff, power);
        }
        break;
    }

    case 2: {
        int coeff, power;
        printf("Enter the coefficient and power for Polynomial 2
(Enter -1 -1 to stop):\n");
        while (1) {
            scanf("%d %d", &coeff, &power);
            if (coeff == -1 && power == -1) {
                break;
            }
            insertTerm(&poly2, coeff, power);
        }
        break;
    }

    case 3:
        printf("Polynomial 1: ");
        displayPolynomial(poly1);
        break;

    case 4:
        printf("Polynomial 2: ");
        displayPolynomial(poly2);
        break;

    case 5: {
        struct Node* result = addPolynomials(poly1, poly2);
        printf("Sum of Polynomials: ");
        displayPolynomial(result);
        cleanup(result);
        break;
    }

    case 6:
        cleanup(poly1);
        cleanup(poly2);
        printf("Exiting the program.\n");
        break;

    default:
        printf("Invalid choice. Please enter a valid option.\n");

```



```

        break;
    }
} while (choice != 6);

return 0;
}

```

Sample Output:

```

Polynomial Menu:
1. Enter Polynomial 1
2. Enter Polynomial 2
3. Display Polynomial 1
4. Display Polynomial 2
5. Add Polynomials
6. Exit
Enter Your Choice: 1
Enter the coefficient and power for Polynomial 1 (Enter -1 -1 to stop):
5 3 2 2 3 0 -1 -1

Polynomial Menu:
1. Enter Polynomial 1
2. Enter Polynomial 2
3. Display Polynomial 1
4. Display Polynomial 2
5. Add Polynomials
6. Exit
Enter Your Choice: 2
Enter the coefficient and power for Polynomial 2 (Enter -1 -1 to stop):
9 3 5 1 6 0 -1 -1

Polynomial Menu:
1. Enter Polynomial 1
2. Enter Polynomial 2
3. Display Polynomial 1
4. Display Polynomial 2
5. Add Polynomials
6. Exit
Enter Your Choice: 3
Polynomial 1: 5x^3 + 2x^2 + 3x^0

Polynomial Menu:
1. Enter Polynomial 1
2. Enter Polynomial 2
3. Display Polynomial 1
4. Display Polynomial 2
5. Add Polynomials
6. Exit
Enter Your Choice: 4
Polynomial 2: 9x^3 + 5x^1 + 6x^0

Polynomial Menu:
1. Enter Polynomial 1
2. Enter Polynomial 2
3. Display Polynomial 1
4. Display Polynomial 2
5. Add Polynomials
6. Exit
Enter Your Choice: 5
Sum of Polynomials: 14x^3 + 2x^2 + 5x^1 + 9x^0

```

Problem - 3.4: Implement doubly Linked List data structure and its operations like insert and delete in the beginning/end and n^{th} position of the list, and display the items stored in the linked list.

Source Code:

```

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

// Structure for Node
struct node{
    struct node*prev;
    int Data;
    struct node*next;
};

// Structure For DMA of node
struct node *Create_Node(){
    return (struct node*)malloc(sizeof(struct node));
}

void traverse(struct node *start){
    struct node *ptr;
    ptr = start;
    while(ptr!=NULL) {

```

```

        printf("%d -> ", ptr->Data);
        ptr = ptr -> next ;
    }
    printf("NULL\n");
}

void traverse_using_End(struct node *end) {

    struct node *ptr = end;
    while(ptr!=NULL) {
        printf("%d -> ", ptr->Data);
        ptr = ptr -> prev ;
    }
    printf("NULL\n");
}

int countNodes(struct node *start) {
    int count = 0;
    struct node *ptr;
    ptr = start;
    while (ptr != NULL) {
        count++;
        ptr = ptr->next;
    }
    printf("No of Nodes are : %d\n", count);
    return count;
}

void insert_begining(struct node **start ,struct node **end, int item){
    struct node *n1 = Create_Node();
    if (n1 == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    n1->Data = item;
    n1->prev = NULL;

    if(*start != NULL){
        n1->next = *start;
        (*start)->prev = n1;
        *start = n1;
    }
    else{
        // If the list is empty
        *start = *end = n1;
        n1->next = NULL;
    }
}

void insert_at_End(struct node **start ,struct node **end, int item){
    struct node *n1 = Create_Node();

    if (n1 == NULL) {
        printf("Memory allocation failed.\n");

```

```

        exit(1);
    }

    n1->Data = item;
    n1->prev = NULL;
    n1->next = NULL;

    if(*start == NULL) *start = *end = n1;
    else{
        n1->prev = *end;
        (*end)->next = n1;
        *end = n1;
    }
}

void insertSpecific(struct node **start, struct node **end, int item, int
pos) {
    struct node *n1 = Create_Node();
    if (n1 == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    n1->Data = item;
    n1->next = NULL;
    n1->prev = NULL;

    if (*start == NULL) *start = *end = n1;

    else if (pos == 0) {
        n1->next = *start;
        (*start)->prev = n1;
        *start = n1;
    }
    else{
        struct node *ptr = *start;
        int currentPos = 0;

        while (ptr->next != NULL && currentPos != pos - 1) {
            ptr = ptr->next;
            currentPos++;
        }

        n1->next = ptr->next;

        if (ptr->next != NULL) ptr->next->prev = n1;
        else *end = n1;

        n1->prev = ptr;
        ptr->next = n1;
    }
}

void del_first(struct node **start, struct node **end) {

```

```

    if(*start == NULL){
        printf("UnderFlow");
        return;
    }

    struct node *ptr;
    ptr = *start;

    *start = (*start)->next;

    if(*start == NULL) *end = NULL;

    else{
        (*start)->prev = NULL;
    }
    free(ptr);
}

void del_end(struct node **start, struct node **end){
    if(*start == NULL){
        printf("UnderFlow");
        return;
    }

    if(*start == *end){
        *start = *end = NULL;
        return;
    }

    struct node *ptr = *end;
    *end = ptr->prev;
    (*end)->next = NULL;
    free(ptr);
}

void del_specific(struct node **start, struct node **end, int pos){
    if (*start == NULL){
        printf("UnderFlow");
        return;
    }

    struct node *ptr = *start;
    int currentPos = 0;

    while (ptr != NULL && currentPos != pos){
        ptr = ptr->next;
        currentPos++;
    }

    if (ptr == NULL){
        printf("Invalid Position");
        return;
    }

    if (ptr == *start){

```

```

        // Case: Deleting the first node
        *start = ptr->next;

        if (*start != NULL) (*start)->prev = NULL;

        else *end = NULL; // Case: List becomes empty after deletion
    }

    else if (ptr == *end){
        // Case: Deleting the last node
        *end = ptr->prev;
        (*end)->next = NULL;
    }

    else{
        // Case: Deleting a node from somewhere in between
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
    }

    free(ptr);
}

int main() {
    struct node *n1, *start = NULL , *end = NULL;
    int item, pos;

    // Insert at the beginning
    insert_begining(&start, &end, 10);
    insert_begining(&start, &end, 20);
    insert_begining(&start, &end, 30);

    printf("List after insert at the beginning: ");
    traverse(start);

    // Insert at the end
    insert_at_End(&start, &end, 40);
    insert_at_End(&start, &end, 50);
    printf("List after insert at the end: ");
    traverse(start);

    // Insert at a specific position
    insertSpecific(&start, &end, 25, 2);
    insertSpecific(&start, &end, 35, 4);
    printf("List after insert at specific positions: ");
    traverse(start);

    // Delete the first node
    del_first(&start, &end);
    printf("List after deleting the first node: ");
    traverse(start);

    // Delete the last node
    del_end(&start, &end);
    printf("List after deleting the last node: ");

```

```

traverse(start);

// Delete a node at a specific position
printf("Enter the position to delete: ");
scanf("%d", &pos);
del_specific(&start, &end, pos);
printf("List after deleting a node at a specific position: ");
traverse(start);

// Count the number of nodes
countNodes(start);

return 0;
}

```

Sample Output:

```

List after insert at the beginning: 30 -> 20 -> 10 -> NULL
List after insert at the end: 30 -> 20 -> 10 -> 40 -> 50 -> NULL
List after insert at specific positions: 30 -> 20 -> 25 -> 10 -> 35 -> 40 -> 50 -> NULL
List after deleting the first node: 20 -> 25 -> 10 -> 35 -> 40 -> 50 -> NULL
List after deleting the last node: 20 -> 25 -> 10 -> 35 -> 40 -> NULL

```

Problem – 3.5: Implement circular Linked List data structure and its operations like insert and delete in the beginning/end and n^{th} position of the list, and display the items stored in the linked list.

Source Code:

```

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

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

struct Node* Create_Node(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed. Unable to create node.\n");
        return NULL;
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Node* getLastNode(struct Node* head) {
    struct Node* lastNode = head;
    while (lastNode->next != head) {
        lastNode = lastNode->next;
    }
}

```

```

        return lastNode;
    }

void insertAtBeginning(struct Node** headRef, int newData) {
    struct Node* newNode = Create_Node(newData);
    if (newNode == NULL) {
        return; // Memory allocation failed
    }

    if (*headRef == NULL) {
        newNode->next = newNode; // Circular link for single node
    } else {
        struct Node* lastNode = getLastNode(*headRef);
        newNode->next = *headRef;
        lastNode->next = newNode;
    }
    *headRef = newNode;
}

void insertAtEnd(struct Node** headRef, int newData) {
    struct Node* newNode = Create_Node(newData);
    if (newNode == NULL) {
        return; // Memory allocation failed
    }

    if (*headRef == NULL) {
        newNode->next = newNode; // Circular link for single node
        *headRef = newNode;
    } else {
        struct Node* lastNode = getLastNode(*headRef);
        newNode->next = *headRef;
        lastNode->next = newNode;
    }
}

void insertAtNthPosition(struct Node** headRef, int newData, int
position) {
    if (position < 1) {
        printf("Invalid position. Position should be >= 1.\n");
        return;
    }

    if (position == 1) {
        insertAtBeginning(headRef, newData);
        return;
    }

    struct Node* newNode = Create_Node(newData);
    if (newNode == NULL) {
        return; // Memory allocation failed
    }
}

```

```

    struct Node* currentNode = *headRef;
    for (int i = 1; i < position - 1 && currentNode != NULL; i++) {
        currentNode = currentNode->next;
    }

    if (currentNode == NULL) {
        printf("Invalid position. Position exceeds the length of the
list.\n");
        free(newNode);
        return;
    }

    newNode->next = currentNode->next;
    currentNode->next = newNode;
}

void deleteFromBeginning(struct Node** headRef) {
    if (*headRef == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }

    if ((*headRef)->next == *headRef) {
        // Single node in the list
        free(*headRef);
        *headRef = NULL;
        return;
    }

    struct Node* lastNode = getLastNode(*headRef);
    struct Node* temp = *headRef;
    *headRef = (*headRef)->next;
    lastNode->next = *headRef;
    free(temp);
}

void deleteFromEnd(struct Node** headRef) {
    if (*headRef == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }

    if ((*headRef)->next == *headRef) {
        // Single node in the list
        free(*headRef);
        *headRef = NULL;
        return;
    }

    struct Node* secondLastNode = *headRef;
    while (secondLastNode->next->next != *headRef) {
        secondLastNode = secondLastNode->next;
    }
}

```



```

        struct Node* lastNode = secondLastNode->next;
        secondLastNode->next = *headRef;
        free(lastNode);
    }

    void deleteFromNthPosition(struct Node** headRef, int position) {
        if (position < 1 || *headRef == NULL) {
            printf("Invalid position or empty list. Nothing to
delete.\n");
            return;
        }

        if (position == 1) {
            deleteFromBeginning(headRef);
            return;
        }

        struct Node* currentNode = *headRef;
        struct Node* prevNode = NULL;
        for (int i = 1; i < position && currentNode != NULL; i++) {
            prevNode = currentNode;
            currentNode = currentNode->next;
        }

        if (currentNode == NULL) {
            printf("Invalid position. Position exceeds the length of the
list.\n");
            return;
        }

        prevNode->next = currentNode->next;
        free(currentNode);
    }

    void displayList(struct Node* head) {
        if (head == NULL) {
            printf("List is empty.\n");
            return;
        }

        struct Node* currentNode = head;
        printf("Circular Linked List: ");
        do {
            printf("%d -> ", currentNode->data);
            currentNode = currentNode->next;
        } while (currentNode != head);
        printf("(head)\n");
    }

    int main() {
        struct Node* head = NULL;

        // Test insertion operations

```

```

insertAtEnd(&head, 10);
displayList(head);
insertAtEnd(&head, 20);
displayList(head);
insertAtBeginning(&head, 5);
displayList(head);
insertAtNthPosition(&head, 15, 3);
displayList(head);

// Test deletion operations
deleteFromEnd(&head);
displayList(head);
deleteFromBeginning(&head);
displayList(head);
deleteFromNthPosition(&head, 2);
displayList(head);

return 0;
}

```

Sample Output:

```

Circular Linked List: 10 -> (head)
Circular Linked List: 10 -> 20 -> (head)
Circular Linked List: 5 -> 10 -> 20 -> (head)
Circular Linked List: 5 -> 10 -> 15 -> 20 -> (head)
Circular Linked List: 5 -> 10 -> 15 -> (head)
Circular Linked List: 10 -> 15 -> (head)
Circular Linked List: 10 -> (head)

```

Experiment-04

Title: Stack Data Structures

Objective: To demonstrate use of arrays and linked list to implement Stack operations and applications of Stack

Problem - 4.1: Using array and functions implement Stack and its operations like push, pop, peek.

Source Code:

```

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

int MAX_SIZE; // Global variable to store the size of the stack

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

```

```

}

void push(int *stack, int *top, int item, int max) {
    if (*top == max - 1) {
        printf("Stack Overflow. Cannot push item.\n");
        return;
    }
    (*top)++;
    stack[*top] = item;
}

void pop(int *stack, int *top) {
    if (*top == -1) {
        printf("Stack Underflow. Cannot pop item.\n");
        return;
    }
    printf("Deleted item: %d\n", stack[*top]);
    (*top)--;
}

void peek(int *stack, int top) {
    if (top == -1) {
        printf("Stack is empty. No top element.\n");
        return;
    }
    printf("Top item is: %d\n", stack[top]);
}

int isEmpty(int *stack, int top) {
    return (top == -1) ? 1 : 0;
}

int isFull(int *stack, int top, int max) {
    return (top == max - 1) ? 1 : 0;
}

int main() {
    int top = -1; // Initialize stack top
    int choice, item;

    printf("Enter the size of the stack: ");
    scanf("%d", &MAX_SIZE); // Read the size of the stack from user input

    int stack[MAX_SIZE]; // Declare stack array based on user input size

    while (1) {
        printf("\nStack Menu:\n");
        printf("1. Display Stack\n");
        printf("2. Insert Element\n");
        printf("3. Delete Element\n");
        printf("4. Top Element\n");
        printf("5. Check If Empty\n");
        printf("6. Check If Full\n");
        printf("7. Exit\n");
    }
}

```

```

printf("Enter Your Choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        display(stack, top);
        break;

    case 2:
        printf("Enter the item to Insert: ");
        scanf("%d", &item);
        push(stack, &top, item, MAX_SIZE);
        display(stack, top);
        break;

    case 3:
        pop(stack, &top);
        display(stack, top);
        break;

    case 4:
        peek(stack, top);
        break;

    case 5:
        if (isEmpty(stack, top)) {
            printf("Stack is empty.\n");
        } else {
            printf("Stack is not empty.\n");
        }
        break;

    case 6:
        if (isFull(stack, top, MAX_SIZE)) {
            printf("Stack is full.\n");
        } else {
            printf("Stack is not full.\n");
        }
        break;

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

    default:
        printf("Invalid choice. Please enter a valid option.\n");
        break;
}

return 0;
}

```

Sample Output:

```
Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 2
Enter the item to Insert: 77
Stack elements: 77 --> Null

Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 2
Enter the item to Insert: 94
Stack elements: 94 --> 77 --> Null

Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 3
Deleted item: 94
Stack elements: 77 --> Null

Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 1
Stack elements: 77 --> Null
```

```
Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 2
Enter the item to Insert: 99
Stack elements: 99 --> 77 --> Null

Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 2
Enter the item to Insert: 88
Stack elements: 88 --> 99 --> 77 --> Null

Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 6
Stack is not full.

Stack Menu:
1. Display Stack
2. Insert Element
3. Delete Element
4. Top Element
5. Check If Empty
6. Check If Full
7. Exit
Enter Your Choice: 5
Stack is not empty
```

Problem - 4.2: Use the stack operations developed in Prob 1 and reverse a string using stack

Source Code:

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

```

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

// Import the stack implementation
#include "DMAStack.c"

void reverse(struct node **top , char *str);

int main() {
    printf("Enter the size of the stack: ");
    scanf("%d", &MAX);

    struct node *top = createNode();
    top->data = 0;
    top->link = NULL;

    char str[MAX];

    printf("Enter a string: ");
    scanf("%s", &str);

    reverse(&top , str);

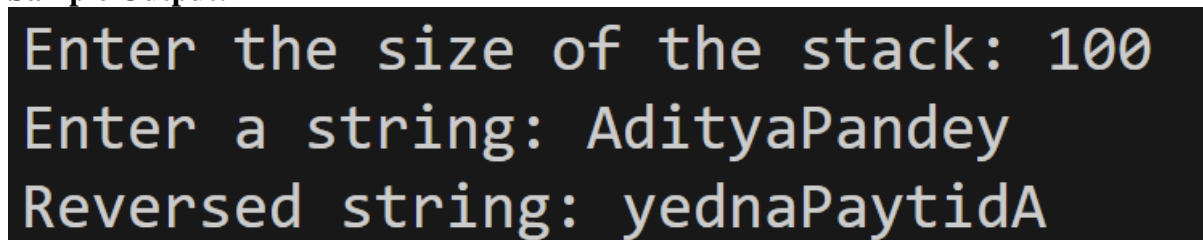
    return 0;
}

void reverse(struct node **top , char *str) {
    for (int i = 0; i < strlen(str); i++) {
        push(top, str[i]);
    }

    printf("Reversed string: ");
    while (isEmpty(*top)!=1) {
        printf("%c", pop(top));
    }
}

```

Sample Output:



```

Enter the size of the stack: 100
Enter a string: AdityaPandey
Reversed string: yednaPaytidA

```

Problem - 4.3: Using array and functions implement two Stacks and its operations (push, pop, peek).

Source Code:

```

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

#define MAX_SIZE 100

typedef struct {

```

```

    int arr[MAX_SIZE];
    int top;
} Stack;

void initialize(Stack *stack) {
    stack->top = -1;
}

void push(Stack *stack, int item) {
    if (stack->top == MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack->arr[++stack->top] = item;
}

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

int peek(Stack *stack) {
    if (stack->top == -1) {
        printf("Stack is empty\n");
        return -1;
    }
    return stack->arr[stack->top];
}

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

int main() {
    Stack stack1, stack2;
    initialize(&stack1);
    initialize(&stack2);

    int choice, stackNum, item;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Stack 1\n");
        printf("2. Stack 2\n");

```

```

printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &stackNum);

if (stackNum == 3) {
    printf("Exiting the program.\n");
    break;
}

Stack *currentStack = (stackNum == 1) ? &stack1 : &stack2;

printf("\nOperations on Stack %d:\n", stackNum);
printf("1. Push\n");
printf("2. Pop\n");
printf("3. Peek\n");
printf("4. Display\n");
printf("Enter operation choice: ");
scanf("%d", &choice);

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

    case 2:
        item = pop(currentStack);
        if (item != -1) {
            printf("Popped item: %d\n", item);
        }
        break;

    case 3:
        item = peek(currentStack);
        if (item != -1) {
            printf("Peeked item: %d\n", item);
        }
        break;

    case 4:
        display(currentStack);
        break;

    default:
        printf("Invalid choice. Please enter a valid option.\n");
}

return 0;
}

```

Sample Output:


```
Menu:
1. Stack 1
2. Stack 2
3. Exit
Enter your choice: 1

Operations on Stack 1:
1. Push
2. Pop
3. Peek
4. Display
Enter operation choice: 1
Enter item to push: 55

Menu:
1. Stack 1
2. Stack 2
3. Exit
Enter your choice: 2

Operations on Stack 2:
1. Push
2. Pop
3. Peek
4. Display
Enter operation choice: 99
Invalid choice. Please enter a valid option.

Menu:
1. Stack 1
2. Stack 2
3. Exit
Enter your choice: 1

Operations on Stack 1:
1. Push
2. Pop
3. Peek
4. Display
Enter operation choice: 4
Stack elements: 55

Menu:
1. Stack 1
2. Stack 2
3. Exit
Enter your choice: 2

Operations on Stack 2:
1. Push
2. Pop
3. Peek
4. Display
Enter operation choice: 4
Stack is empty

Menu:
1. Stack 1
2. Stack 2
3. Exit
Enter your choice: 1

Operations on Stack 1:
1. Push
2. Pop
3. Peek
4. Display
Enter operation choice: 1
Enter item to push: 70

Menu:
1. Stack 1
2. Stack 2
3. Exit
Enter your choice: 1

Operations on Stack 1:
1. Push
2. Pop
3. Peek
4. Display
Enter operation choice: 4
Stack elements: 70 55
```

Problem – 4.4: Implement stack operations using linear linked list.

Source Code:

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

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

typedef struct Stack {
    Node* top;
} Stack;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```

void initialize(Stack* stack) {
    stack->top = NULL;
}

int isEmpty(Stack* stack) {
    return (stack->top == NULL);
}

void push(Stack* stack, int data) {
    Node* newNode = createNode(data);
    newNode->next = stack->top;
    stack->top = newNode;
}

int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow: Cannot pop from empty stack\n");
        exit(EXIT_FAILURE);
    }
    Node* temp = stack->top;
    int data = temp->data;
    stack->top = temp->next;
    free(temp);
    return data;
}

int peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        exit(EXIT_FAILURE);
    }
    return stack->top->data;
}

void display(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements: ");
    Node* current = stack->top;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Stack stack;
    initialize(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

```

```

display(&stack);

printf("Top element: %d\n", peek(&stack));

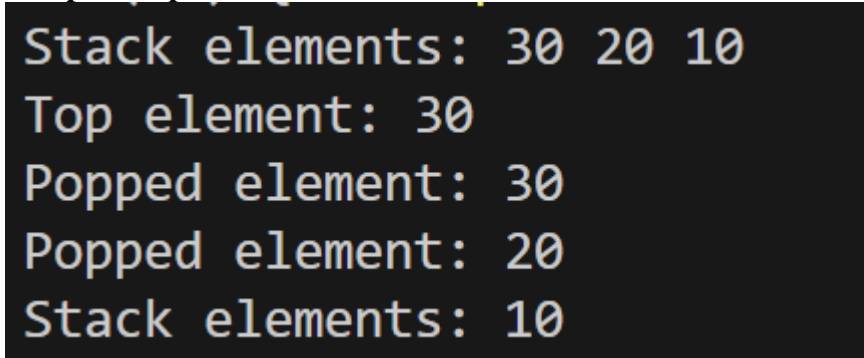
printf("Popped element: %d\n", pop(&stack));
printf("Popped element: %d\n", pop(&stack));

display(&stack);

return 0;
}

```

Sample Output:



```

Stack elements: 30 20 10
Top element: 30
Popped element: 30
Popped element: 20
Stack elements: 10

```

Experiment-05

Title: Stack Data Structures

Objective: To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.1: Using circular array and functions implement Queue data structure and its operations like insert, delete.

Source Code:

```

#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5

typedef struct Queue {
    int items[MAX_SIZE];
    int front, rear;
    int size;
} Queue;

void initializeQueue(Queue *queue) {
    queue->front = 0;
    queue->rear = -1;
    queue->size = 0;
}

int isEmpty(Queue *queue) {
    return (queue->size == 0);
}

int isFull(Queue *queue) {
    return (queue->size == MAX_SIZE);
}

```

```

}

void enqueue(Queue *queue, int item) {
    if (isFull(queue)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    queue->rear = (queue->rear + 1) % MAX_SIZE;
    queue->items[queue->rear] = item;
    queue->size++;
    printf("Enqueued element: %d\n", item);
}

int dequeue(Queue *queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
        exit(EXIT_FAILURE);
    }
    int dequeuedItem = queue->items[queue->front];
    queue->front = (queue->front + 1) % MAX_SIZE;
    queue->size--;
    return dequeuedItem;
}

void displayQueue(Queue *queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    int i;
    for (i = 0; i < queue->size; i++) {
        printf("%d ", queue->items[(queue->front + i) % MAX_SIZE]);
    }
    printf("\n");
}

int main() {
    Queue queue;
    initializeQueue(&queue);
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);
    displayQueue(&queue);
    int dequeuedItem = dequeue(&queue);
    printf("Dequeued element: %d\n", dequeuedItem);

    displayQueue(&queue);

    return 0;
}

```

Sample Output:

```
Enqueued element: 10
Enqueued element: 20
Enqueued element: 30
Queue elements: 10 20 30
Dequeued element: 10
Queue elements: 20 30
```

Problem - 5.2: Implement Queue data structure using linked list and its operations (Enqueue, Dequeue, Display).

Source Code:

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

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

typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;

void initializeQueue(Queue* queue) {
    queue->front = NULL;
    queue->rear = NULL;
}

int isEmpty(Queue* queue) {
    return (queue->front == NULL);
}

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void enqueue(Queue* queue, int data) {
    Node* newNode = createNode(data);
    if (isEmpty(queue)) {
        queue->front = newNode;
    } else {
        queue->rear->next = newNode;
    }
}
```

```

    }
    queue->rear = newNode;
}

int dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
        exit(EXIT_FAILURE);
    }
    Node* frontNode = queue->front;
    int data = frontNode->data;
    queue->front = queue->front->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    free(frontNode);
    return data;
}

void displayQueue(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    Node* current = queue->front;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue queue;
    initializeQueue(&queue);
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);
    displayQueue(&queue);
    int dequeuedItem = dequeue(&queue);
    printf("Dequeued element: %d\n", dequeuedItem);
    displayQueue(&queue);
    return 0;
}

```

Sample Output:

```

Queue elements: 10 20 30
Dequeued element: 10
Queue elements: 20 30

```

Problem - 5.3:

Source Code:

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

#define SIZE 100

typedef struct {
    int top;
    char arr[SIZE];
} Stack;

typedef struct {
    int front, rear;
    char arr[SIZE];
} Queue;

void push(Stack* stack, char c) {
    if (stack->top == SIZE - 1) {
        printf("Stack overflow\n");
        exit(EXIT_FAILURE);
    }
    stack->arr[++stack->top] = c;
}

char pop(Stack* stack) {
    if (stack->top == -1) {
        printf("Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return stack->arr[stack->top--];
}

void enqueue(Queue* queue, char c) {
    if (queue->rear == SIZE - 1) {
        printf("Queue overflow\n");
        exit(EXIT_FAILURE);
    }
    queue->arr[++queue->rear] = c;
}

char dequeue(Queue* queue) {
    if (queue->front == queue->rear) {
        printf("Queue underflow\n");
        exit(EXIT_FAILURE);
    }
    return queue->arr[++queue->front];
}

int isPalindrome(char* str) {
    int len = strlen(str);
```

```

Stack stack;
stack.top = -1;
Queue queue;
queue.front = -1;
queue.rear = -1;

for (int i = 0; i < len; i++) {
    char c = str[i];
    push(&stack, c);
    enqueue(&queue, c);
}

while (queue.front != queue.rear) {
    char stackChar = pop(&stack);
    char queueChar = dequeue(&queue);
    if (stackChar != queueChar) {
        return 0;
    }
}
return 1;
}

int main() {
    char str[SIZE];
    printf("Enter the string: ");
    scanf("%s", str);

    if (isPalindrome(str)) {
        printf("Yes, \"%s\" is a palindrome.\n", str);
    } else {
        printf("No, \"%s\" is not a palindrome.\n", str);
    }

    return 0;
}

```

Sample Output:

Enter the string: NAMAN Yes, "NAMAN" is a palindrome.	Enter the string: ADIDAS No, "ADIDAS" is not a palindrome.
--	---

Problem - 5.4: Implement Double Ended Queue data structure using linked list.

- Input Restricted
- Output Restricted

Source Code:

```

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

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

```



```

Node* newNode(int data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->next = node->prev = NULL;
    return node;
}

void insertAtBeginning(Node** head, int data) {
    Node* node = newNode(data);
    if (*head == NULL) {
        *head = node;
        return;
    }
    node->next = *head;
    (*head)->prev = node;
    *head = node;
}

void insertAtEnd(Node** head, int data) {
    Node* node = newNode(data);
    if (*head == NULL) {
        *head = node;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = node;
    node->prev = temp;
}

void deleteFromBeginning(Node** head) {
    if (*head == NULL) {
        printf("Deque is empty!\n");
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;
    if (*head != NULL) {
        (*head)->prev = NULL;
    }
    free(temp);
}

void deleteFromEnd(Node** head) {
    if (*head == NULL) {
        printf("Deque is empty!\n");
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
}

```

```

    if (temp->prev != NULL) {
        temp->prev->next = NULL;
    } else {
        *head = NULL;
    }
    free(temp);
}

void display(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    Node* head = NULL;
    printf("Input-restricted deque:\n");
    insertAtBeginning(&head, 1);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 3);
    display(head);
    deleteFromEnd(&head);
    display(head);

    printf("Output-restricted deque:\n");
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);
    insertAtEnd(&head, 6);
    display(head);
    deleteFromBeginning(&head);
    display(head);

    return 0;
}

```

Sample Output:

```

Input-restricted deque:
3 2 1
3 2
Output-restricted deque:
3 2 4 5 6
2 4 5 6

```

Problem - 5.5: Implement Priority Queue using array where the minimum element is having highest priority

Source Code:

```

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

```

```

struct Element {
    int item;
    int priority;
};

struct PriorityQueue {
    struct Element* queue;
    int size;
    int capacity;
};

struct PriorityQueue* createPriorityQueue(int capacity) {
    struct PriorityQueue* pq = (struct PriorityQueue*)malloc(sizeof(struct
PriorityQueue));
    pq->queue = (struct Element*)malloc(capacity * sizeof(struct
Element));
    pq->size = 0;
    pq->capacity = capacity;
    return pq;
}

void enqueue(struct PriorityQueue* pq, int item, int priority) {
    if (pq->size == pq->capacity) {
        printf("Priority queue is full. Cannot enqueue.\n");
        return;
    }
    pq->queue[pq->size].item = item;
    pq->queue[pq->size].priority = priority;
    pq->size++;
}

int dequeue(struct PriorityQueue* pq) {
    if (pq->size == 0) {
        printf("Priority queue is empty. Cannot dequeue.\n");
        exit(1);
    }
    int minIndex = 0;
    for (int i = 1; i < pq->size; i++) {
        if (pq->queue[i].priority < pq->queue[minIndex].priority) {
            minIndex = i;
        }
    }
    int item = pq->queue[minIndex].item;
    pq->queue[minIndex] = pq->queue[pq->size - 1];
    pq->size--;
    return item;
}

int isEmpty(struct PriorityQueue* pq) {
    return pq->size == 0;
}

int peek(struct PriorityQueue* pq) {
    if (isEmpty(pq)) {

```

```

        printf("Priority queue is empty.\n");
        exit(1);
    }
    int minIndex = 0;
    for (int i = 1; i < pq->size; i++) {
        if (pq->queue[i].priority < pq->queue[minIndex].priority) {
            minIndex = i;
        }
    }
    return pq->queue[minIndex].item;
}

void destroyPriorityQueue(struct PriorityQueue* pq) {
    free(pq->queue);
    free(pq);
}

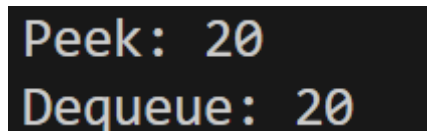
int main() {
    struct PriorityQueue* pq = createPriorityQueue(10);
    enqueue(pq, 10, 3);
    enqueue(pq, 20, 1);
    enqueue(pq, 30, 2);

    printf("Peek: %d\n", peek(pq)); // Should print 20
    printf("Dequeue: %d\n", dequeue(pq)); // Should print 20

    destroyPriorityQueue(pq);
    return 0;
}

```

Sample Output:



```

Peek: 20
Dequeue: 20

```

Problem - 5.6: Implement Priority Queue using Linked list where the priority is associated with each element.

Source Code:

```

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

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

struct PriorityQueue {
    struct Node* head;
};

struct Node* createNode(int item, int priority) {

```

```

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode) {
        newNode->data = item;
        newNode->prio = priority;
        newNode->next = NULL;
    }
    return newNode;
}

struct PriorityQueue* createPQ() {
    struct PriorityQueue* pq = (struct PriorityQueue*)malloc(sizeof(struct
PriorityQueue));
    if (pq) {
        pq->head = NULL;
    }
    return pq;
}

void enqueue(struct PriorityQueue* pq, int item, int priority) {
    struct Node* newNode = createNode(item, priority);
    if (!pq->head || priority < pq->head->prio) {
        newNode->next = pq->head;
        pq->head = newNode;
    } else {
        struct Node* curr = pq->head;
        while (curr->next && curr->next->prio <= priority) {
            curr = curr->next;
        }
        newNode->next = curr->next;
        curr->next = newNode;
    }
}

int dequeue(struct PriorityQueue* pq) {
    if (!pq->head) {
        printf("Empty PQ. Cannot dequeue.\n");
        exit(1);
    }
    int item = pq->head->data;
    struct Node* temp = pq->head;
    pq->head = pq->head->next;
    free(temp);
    return item;
}

int isEmpty(struct PriorityQueue* pq) {
    return !pq->head;
}

int peek(struct PriorityQueue* pq) {
    if (!pq->head) {
        printf("Empty PQ.\n");
        exit(1);
    }
    return pq->head->data;
}

```

```

}

void destroyPQ(struct PriorityQueue* pq) {
    while (pq->head) {
        struct Node* temp = pq->head;
        pq->head = pq->head->next;
        free(temp);
    }
    free(pq);
}

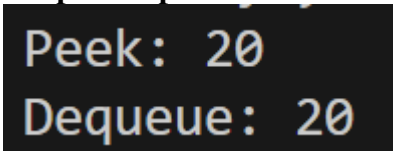
int main() {
    struct PriorityQueue* pq = createPQ();
    enqueue(pq, 10, 3);
    enqueue(pq, 20, 1);
    enqueue(pq, 30, 2);

    printf("Peek: %d\n", peek(pq)); // Should print 20
    printf("Dequeue: %d\n", dequeue(pq)); // Should print 20

    destroyPQ(pq);
    return 0;
}

```

Sample Output:



```

Peek: 20
Dequeue: 20

```

Experiment-06

Title: Trees

Objective: To demonstrate the creation of a binary tree using arrays/linked lists and working with tree traversal and heap sorting algorithms.

Problem - 6.1: Create a binary tree using an array/linked List. Write the functions to perform Preorder, Inorder, Postorder and Level- order Traversal of constructed tree.

Source Code:

```

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

struct node {
    int data;
    struct node* left;
    struct node* right;
};

struct node* CreateNode(int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    if (newNode != NULL) {
        newNode->data = data;
        newNode->left = NULL;
        newNode->right = NULL;
    }
}

```

```

        return newNode;
    }

void addBSTnode(struct node** root, int data) {
    struct node* newNode = CreateNode(data);
    if (*root == NULL) {
        *root = newNode;
        return;
    }

    struct node* current = *root;
    struct node* parent = NULL;

    while (current != NULL) {
        parent = current;
        if (data < current->data) {
            current = current->left;
        } else if (data > current->data) {
            current = current->right;
        } else {
            free(newNode); // Free the unused node (duplicate element)
            return;
        }
    }

    if (data < parent->data) {
        parent->left = newNode;
    } else {
        parent->right = newNode;
    }
}

void InOrderTraversal(struct node* root) {
    if (root != NULL) {
        InOrderTraversal(root->left);
        printf("%d ", root->data);
        InOrderTraversal(root->right);
    }
}

void PreOrderTraversal(struct node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        PreOrderTraversal(root->left);
        PreOrderTraversal(root->right);
    }
}

void PostOrderTraversal(struct node* root) {
    if (root != NULL) {
        PostOrderTraversal(root->left);
        PostOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}

```

```

int main() {
    struct node* root = NULL;

    addBSTnode(&root, 10);
    addBSTnode(&root, 5);
    addBSTnode(&root, 3);
    addBSTnode(&root, 2);
    addBSTnode(&root, 4);
    addBSTnode(&root, 7);
    addBSTnode(&root, 6);
    addBSTnode(&root, 8);
    addBSTnode(&root, 9);
    addBSTnode(&root, 15);
    addBSTnode(&root, 12);
    addBSTnode(&root, 11);
    addBSTnode(&root, 13);
    addBSTnode(&root, 14);
    addBSTnode(&root, 18);
    addBSTnode(&root, 17);
    addBSTnode(&root, 19);

    printf("Inorder Traversal: ");
    InOrderTraversal(root);
    printf("\n");

    printf("Preorder Traversal: ");
    PreOrderTraversal(root);
    printf("\n");

    printf("Postorder Traversal: ");
    PostOrderTraversal(root);
    printf("\n");

    return 0;
}

```

Sample Output:

```

Inorder Traversal: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19
Preorder Traversal: 10 5 3 2 4 7 6 8 9 15 12 11 13 14 18 17 19
Postorder Traversal: 2 4 3 6 9 8 7 5 11 14 13 12 17 19 18 15 10

```

Problem - 6.2:

Source Code:

```

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

struct node {
    int data;
    struct node* left;
    struct node* right;
}

```



```

};

struct node* CreateNode(int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    if (newNode != NULL) {
        newNode->data = data;
        newNode->left = NULL;
        newNode->right = NULL;
    }
    return newNode;
}

void addBSTnode(struct node** root, int data) {
    struct node* newNode = CreateNode(data);
    if (*root == NULL) {
        *root = newNode;
        return;
    }

    struct node* current = *root;
    struct node* parent = NULL;

    while (current != NULL) {
        parent = current;
        if (data < current->data) {
            current = current->left;
        } else if (data > current->data) {
            current = current->right;
        } else {
            free(newNode); // Free the unused node (duplicate element)
            return;
        }
    }

    if (data < parent->data) {
        parent->left = newNode;
    } else {
        parent->right = newNode;
    }
}

struct node* Search(struct node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return Search(root->left, data);
    }
    return Search(root->right, data);
}

struct node* findMinNode(struct node* node) {
    struct node* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
}

```

```

    }
    return current;
}

struct node* deleteNode(struct node* root, int data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root);
            return temp;
        }
        struct node* temp = findMinNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void InOrderTraversal(struct node* root) {
    if (root != NULL) {
        InOrderTraversal(root->left);
        printf("%d ", root->data);
        InOrderTraversal(root->right);
    }
}

void PreOrderTraversal(struct node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        PreOrderTraversal(root->left);
        PreOrderTraversal(root->right);
    }
}

void PostOrderTraversal(struct node* root) {
    if (root != NULL) {
        PostOrderTraversal(root->left);
        PostOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}

void displayMenu() {

```

```

printf("\nBinary Search Tree Operations Menu\n");
printf("1. Insert a node\n");
printf("2. Search for a node\n");
printf("3. Delete a node\n");
printf("4. Inorder Traversal\n");
printf("5. Preorder Traversal\n");
printf("6. Postorder Traversal\n");
printf("7. Exit\n");
printf("Enter your choice: ");
}

int main() {
    struct node* root = NULL;
    int choice, data;

    do {
        displayMenu();
        scanf("%d", &choice);

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

            case 2:
                printf("Enter value to search: ");
                scanf("%d", &data);
                if (Search(root, data) != NULL) {
                    printf("%d found in the tree.\n", data);
                } else {
                    printf("%d not found in the tree.\n", data);
                }
                break;

            case 3:
                printf("Enter value to delete: ");
                scanf("%d", &data);
                root = deleteNode(root, data);
                break;

            case 4:
                printf("Inorder Traversal: ");
                InOrderTraversal(root);
                printf("\n");
                break;

            case 5:
                printf("Preorder Traversal: ");
                PreOrderTraversal(root);
                printf("\n");
                break;

            case 6:

```

```

        printf("Postorder Traversal: ");
        PostOrderTraversal(root);
        printf("\n");
        break;

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

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

    } while (choice != 7);

    return 0;
}

```

Sample Output:

<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 1 Enter value to insert: 55 </pre>	<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 1 Enter value to insert: 23 </pre>	<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 6 Postorder Traversal: 23 44 90 55 </pre>
<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 1 Enter value to insert: 44 </pre>	<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 4 Inorder Traversal: 23 44 55 90 </pre>	<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 2 Enter value to search: 90 90 found in the tree. </pre>
<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 1 Enter value to insert: 90 </pre>	<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 5 Preorder Traversal: 55 44 23 90 </pre>	<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 3 Enter value to delete: 90 </pre>
		<pre> Binary Search Tree Operations Menu 1. Insert a node 2. Search for a node 3. Delete a node 4. Inorder Traversal 5. Preorder Traversal 6. Postorder Traversal 7. Exit Enter your choice: 4 Inorder Traversal: 23 44 55 </pre>

Problem - 6.3: Write a program to perform the following operations

- a. Find Minimum Element
- b. Find Maximum Element

Source Code:

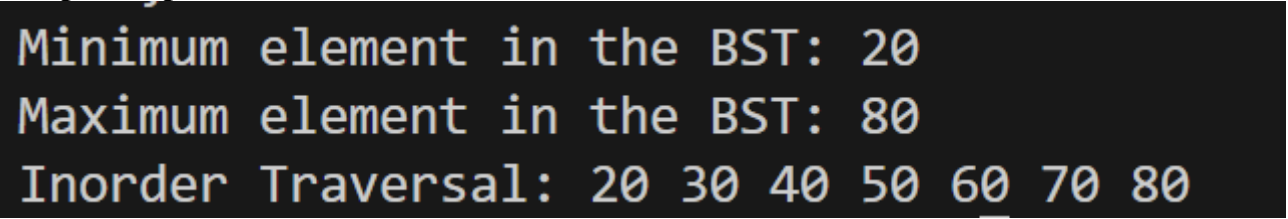
```
struct node* findMin(struct node* root) {
    if (root == NULL) {
        return NULL; // Tree is empty
    }

    struct node* current = root;
    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}

struct node* findMax(struct node* root) {
    if (root == NULL) {
        return NULL; // Tree is empty
    }

    struct node* current = root;
    while (current->right != NULL) {
        current = current->right;
    }
    return current;
}
```

Sample Output:



```
Minimum element in the BST: 20
Maximum element in the BST: 80
Inorder Traversal: 20 30 40 50 60 70 80
```

Problem - 6.4: Write the program that reads the random sequence of integers and prints the sorted form of given data (ascending Order) using Binary Search Tree.

Source Code:

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

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* createNode(int data) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
    }
}
```

```

        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct TreeNode* insertNode(struct TreeNode* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}

void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

int main() {
    struct TreeNode* root = NULL;
    int num;

    printf("Enter integers (EOF or non-integer to end input):\n");
    while (1) {
        if (scanf("%d", &num) == 1) {
            root = insertNode(root, num);
        } else {
            break; // Exit loop if non-integer or EOF is encountered
        }
    }
    printf("Sorted order (ascending):\n");
    inorderTraversal(root);
    printf("\n");

    return 0;
}

```

Sample Output:

```

Enter integers (EOF or non-integer to end input):
99 33 21 90 43 67 12 3 end
Sorted order (ascending):
3 12 21 33 43 67 90 99

```

Experiment-07

Title: Searching and Sorting

Objective: To implement various searching & sorting algorithms.

Problem - 7.1: Read the numbers from a text file sort them into an array using '*Insertion Sort*' algorithm and write back in another text file.

Source Code:

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

#define MAX_SIZE 100

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
    FILE *inputFile, *outputFile;
    int numbers[MAX_SIZE];
    int num, count = 0;

    inputFile = fopen("input.txt", "r");
    if (inputFile == NULL) {
        perror("Error opening input.txt");
        return 1;
    }

    while (fscanf(inputFile, "%d", &num) == 1) {
        if (count >= MAX_SIZE) {
            printf("Exceeded maximum size of array. Cannot process all
numbers.\n");
            break;
        }
        numbers[count++] = num;
    }

    fclose(inputFile);

    insertionSort(numbers, count);

    outputFile = fopen("output.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output.txt");
    }
}
```

```

        return 1;
    }

    for (int i = 0; i < count; i++) {
        fprintf(outputFile, "%d ", numbers[i]);
    }
    fprintf(outputFile, "\n");

    fclose(outputFile);

    printf("Numbers have been sorted and written to output.txt\n");

    return 0;
}

```

Sample Output:

```

99 33 21 90 43 67 12 3
3 12 21 33 43 67 90 99

```

Problem - 7.2: Read the numbers from a text file sort them into an array using ‘*Bubble Sort*’ algorithm and write back in another text file.

Source Code:

```

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

#define MAX_SIZE 100

void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    FILE *inputFile, *outputFile;
    int numbers[MAX_SIZE];
    int num, count = 0;

    // Open input file for reading
    inputFile = fopen("input.txt", "r");
    if (inputFile == NULL) {

```



```

        perror("Error opening input.txt");
        return 1;
    }

    // Read integers from input file into array
    while (fscanf(inputFile, "%d", &num) == 1) {
        if (count >= MAX_SIZE) {
            printf("Exceeded maximum size of array. Cannot process all
numbers.\n");
            break;
        }
        numbers[count++] = num;
    }

    fclose(inputFile);

    // Sort the array using Bubble Sort
    bubbleSort(numbers, count);

    // Open output file for writing
    outputFile = fopen("output.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output.txt");
        return 1;
    }

    // Write sorted numbers to output file
    for (int i = 0; i < count; i++) {
        fprintf(outputFile, "%d ", numbers[i]);
    }
    fprintf(outputFile, "\n");

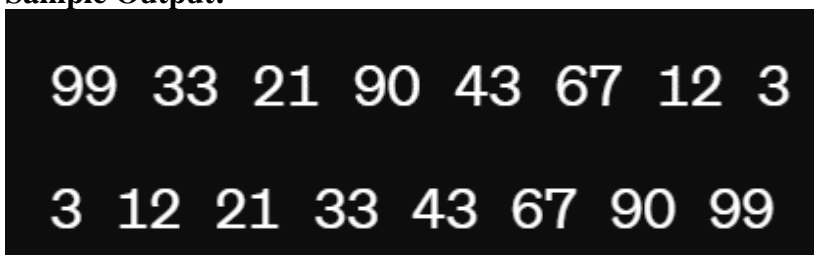
    fclose(outputFile);

    printf("Numbers have been sorted and written to output.txt\n");

    return 0;
}

```

Sample Output:



```

99 33 21 90 43 67 12 3
3 12 21 33 43 67 90 99

```

Problem - 7.3: Read the numbers from a text file and write a function to search an element using linear search.

Source Code:

```

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

```

```

#define MAX_SIZE 100

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

int main() {
    FILE *file;
    int numbers[MAX_SIZE];
    int num, count = 0;
    char filename[] = "numbers.txt";

    file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    while (fscanf(file, "%d", &num) == 1) {
        if (count >= MAX_SIZE) {
            printf("Exceeded maximum size of array. Cannot process all
numbers.\n");
            break;
        }
        numbers[count++] = num;
    }

    fclose(file);

    printf("Numbers read from file:\n");
    for (int i = 0; i < count; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

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

    int index = linearSearch(numbers, count, key);
    if (index != -1) {
        printf("%d found at index %d\n", key, index);
    } else {
        printf("%d not found in the list\n", key);
    }

    return 0;
}

```

Sample Output:

```
Numbers read from file:
99 33 21 90 43 67 12 3
Enter a number to search: 67
67 found at index 5
```

Problem - 7.4: Read the numbers from a text file sort them into an array using ‘*Selection Sort*’ algorithm and write back in another text file.

Source Code:

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

#define MAX_SIZE 100

// Function to perform Selection Sort on an array of integers
void selectionSort(int arr[], int n) {
    int i, j, min_idx, temp;
    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        // Swap the found minimum element with the first element of the
        // unsorted part
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    FILE *inputFile, *outputFile;
    int numbers[MAX_SIZE];
    int num, count = 0;

    // Open input file for reading
    inputFile = fopen("input.txt", "r");
    if (inputFile == NULL) {
        perror("Error opening input.txt");
        return 1;
    }

    // Read integers from input file into array
    while (fscanf(inputFile, "%d", &num) == 1) {
        if (count >= MAX_SIZE) {
            printf("Exceeded maximum size of array. Cannot process all
numbers.\n");
            break;
        }
        numbers[count++] = num;
    }
}
```

```

}

fclose(inputFile);

// Sort the array using Selection Sort
selectionSort(numbers, count);

// Open output file for writing
outputFile = fopen("output.txt", "w");
if (outputFile == NULL) {
    perror("Error opening output.txt");
    return 1;
}

// Write sorted numbers to output file
for (int i = 0; i < count; i++) {
    fprintf(outputFile, "%d ", numbers[i]);
}
fprintf(outputFile, "\n");

fclose(outputFile);

printf("Numbers have been sorted and written to output.txt\n");

return 0;
}

```

Sample Output:



```

45 12 78 34 56 23 91 8
8 12 23 34 45 56 78 91

```

Problem - 7.5: Read the numbers from a text file where numbers are stored in random manner. Write a program to search an element in the data using Binary Search. (Note: we require sorted data for applying binary Search Algorithm.

Source Code:

```

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

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

```

```

        return -1;
    }

void readAndSortNumbers(const char *filename, int arr[], int *size) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }
    *size = 0;
    int num;
    while (fscanf(file, "%d", &num) == 1) {
        arr[*size] = num;
        (*size)++;
    }
    fclose(file);
    qsort(arr, *size, sizeof(int), compareIntegers);
}

int compareIntegers(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int main() {
    const char *filename = "numbers.txt";
    int arr[1000];
    int size;

    readAndSortNumbers(filename, arr, &size);

    int target;
    printf("Enter a number to search: ");
    scanf("%d", &target);

    int index = binarySearch(arr, 0, size - 1, target);

    if (index != -1) {
        printf("%d found at index %d.\n", target, index);
    } else {
        printf("%d not found in the array.\n", target);
    }
    return 0;
}

```

Sample Output:

```

Numbers read from file:
99 33 21 90 43 67 12 3
Enter a number to search: 67
67 found at index 5.

```