**Data Structures using C**

**Lab File**

Submitted to:

**AMITY UNIVERSITY UTTTAR PARDESH**

****

**In partial fulfilment of the requirements for the award of the degree of**

**Bachelor of technology**

**In**

Computer Science & Engineering

By

*Faizan Ahmed Syed*

A23055223216

CSE 4-X Semester 3

Submitted to:

*Dr. Smriti Sehgal*

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY**

**AMITY UNIVERSITY UTTAR PARDESH**

**NOIDA (U.P.)**

**LIST OF EXPERIMENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Experiment | Date | Page No. | Sign |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Experiment | Date | Page No. | Sign |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Experiment | Date | Page No. | Sign |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Lab 1**

**Program:**

Write a menu driven program for the insertion of elements in the beginning, middle, and the end of a 1-D array.

**Source Code:**

#include<stdio.h>

void beginning(int new\_element, int n, int arr[]) {

    for(int i = n; i > 0; i--) {

        arr[i] = arr[i - 1];

    }

    arr[0] = new\_element;

    printf("New array: ");

    for(int i = 0; i <= n; i++){

        printf("%d ", arr[i]);

    } printf("\n\n");

}

void middle(int new\_element, int arr[], int position, int n) {

    for(int i = n; i >= position; i--) {

        arr[i] = arr[i - 1];

        }

        arr[position] = new\_element;

    printf("New array: ");

    for(int i = 0; i <= n; i++)

        printf("%d ", arr[i]);

     printf("\n\n");

}

void end(int new\_element, int n, int arr[]) {

    arr[n] = new\_element;

    printf("New array: ");

    for(int i = 0; i <= n; i++){

        printf("%d ", arr[i]);

    } printf("\n\n");

}

int main() {

    int arr[100];

    int n;

    int option;

    int new\_element;

    int position;

    char repeat;

    printf("\n -> Insertion Option Menu:\n");

    printf("\nSelect the number for the task you want to perform.\n");

    printf("1- Beginning Insertion.\n2- Middle Insertion.\n3- End Insertion.\n");

    printf("\nCreate an array: \n");

    printf("Enter number of elements: ");

    scanf("%d", &n);

    printf("\nEnter the %d elements of the array: \n", n);

    for(int i = 0; i < n; i++){

        printf("Element %d: ", i);

        scanf("%d", &arr[i]);

    }

    printf("The array is: \n");

    for(int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    } printf("\n\n");

    do {

        printf("Select the task (1-3): ");

        scanf("%d", &option);

        printf("\n");

        switch(option) {

                case 1:

                    printf("Enter value to be inserted: ");

                    scanf("%d", &new\_element);

                    beginning(new\_element, n, arr);

                    n = n + 1;

                    break;

                case 2:

                    printf("Enter value to be inserted: ");

                    scanf("%d", &new\_element);

                    printf("Which position do you want to insert at: ");

                    scanf("%d", &position);

                    if(position < 0 || position > n) {

                        printf("Invalid position...\n");

                        n = n + 1;

                        break;

                    }

                    middle(new\_element, arr, position, n);

                    break;

                case 3:

                    printf("Enter value to be inserted: ");

                    scanf("%d", &new\_element);

                    n = n + 1;

                    end(new\_element, n, arr);

                    break;

                default:

                    printf("Selected option does not exist.\n");

                    printf("Please choose from 1-3 only...\n\n");

                    break;

            }

            printf("Do you want to continue (Y/N): ");

            scanf(" %c", &repeat);

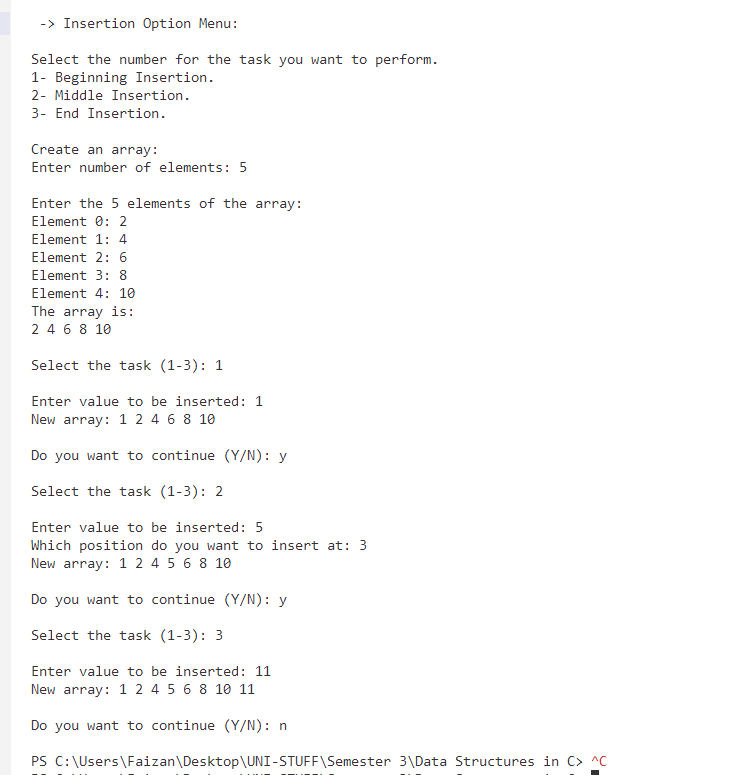
            printf("\n");

    }

    while(repeat == 'Y' || repeat == 'y');

    return 0;

}

**Output:** 

**Lab 2**

**Program:**

Write a menu driven program for the deletion of elements from the beginning middle and end of a 1-D array.

**Source Code:**

#include<stdio.h>

 void beginning(int n, int arr[]) {

    for(int i = 0; i < n - 1; i++) {

        arr[i] = arr[i + 1];

    }

    n = n - 1;

    printf("New array: ");

    for(int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    } printf("\n\n");

}

void middle(int arr[], int position, int n) {

    for(int i = position; i <= n - 1; i++) {

        arr[i] = arr[i + 1];

        }

        n = n - 1;

    printf("New array: ");

    for(int i = 0; i < n; i++)

        printf("%d ", arr[i]);

     printf("\n\n");

}

void end(int n, int arr[]) {

    n = n - 1;

    printf("New array: ");

    for(int i = 0; i < n; i++){

        printf("%d ", arr[i]);

    } printf("\n\n");

}

 int main() {

    int arr[100];

    int n;

    int option;

    int position;

    char repeat;

    printf("\n -> Deletion Option Menu:\n");

    printf("\nSelect the number for the task you want to perform.\n");

    printf("1- First Element Deletion.\n2- Middle Element Deletion.\n3- Last Element Deletion.\n");

    printf("\nCreate an array: \n");

    printf("Enter number of elements: ");

    scanf("%d", &n);

    printf("\nEnter the %d elements of the array: \n", n);

    for(int i = 0; i < n; i++){

        printf("Element %d: ", i);

        scanf("%d", &arr[i]);

    }

    printf("The array is: \n");

    for(int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    } printf("\n\n");

    do {

        printf("Select the task (1-3): ");

        scanf("%d", &option);

        printf("\n");

        switch(option) {

                case 1:

                    beginning(n, arr);

                    n = n - 1;

                    break;

                case 2:

                    printf("Which position element do you want to delete: ");

                    scanf("%d", &position);

                    if(position < 0 || position > n) {

                        printf("Invalid position...\n");

                        break;

                    }

                    middle(arr, position, n);

                    n = n - 1;

                    break;

                case 3:

                    end(n, arr);

                    break;

                default:

                    printf("Selected option does not exist.\n");

                    printf("Please choose from 1-3 only...\n\n");

                    break;

            }

            printf("Do you want to continue (Y/N): ");

            scanf(" %c", &repeat);

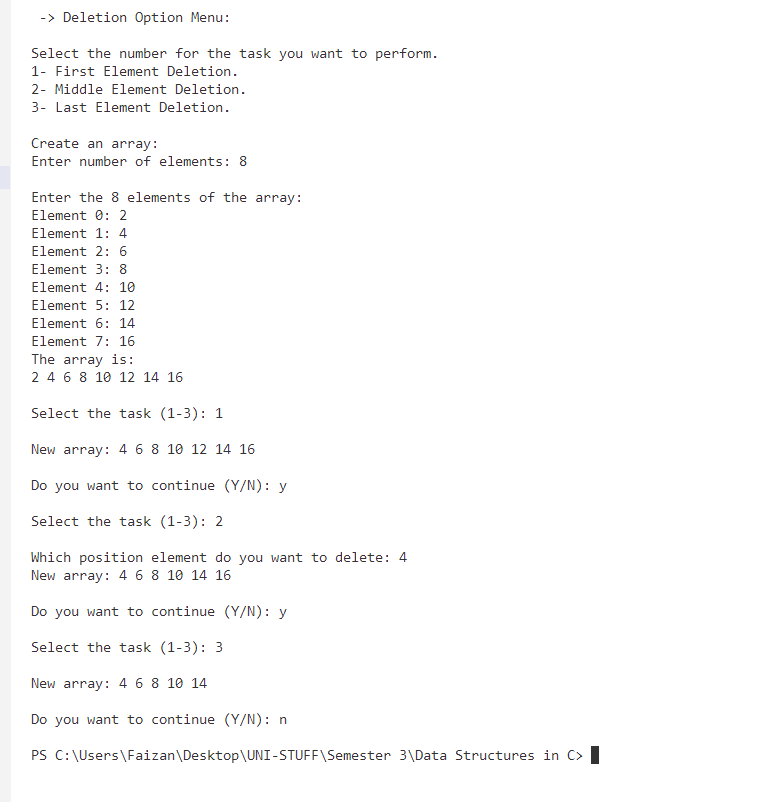
            printf("\n");

    }

    while(repeat == 'Y' || repeat == 'y');

    return 0;

 }

**Output:** ****

**Lab 3**

**Program:**

Write a program to perform linear search and binary search on a 1-D array.

**Source Code:**

#include<stdio.h>

int linear\_search(int arr[], int n, int val) {

    for(int i = 0; i < n; i++) {

        if(arr[i] == val) {

            return i;

        }

    }

    return  -1;

}

int binary\_search(int arr[], int low, int high, int val) {

    if(high >= low) {

        int mid = low + (high - low) / 2;

        if(arr[mid] == val)

            return mid;

        if(arr[mid] > val)

            return binary\_search(arr, low, mid-1, val);

        return binary\_search(arr, mid+1, high, val);

    }

    return -1;

}

int main() {

    int arr[100], n, val, index, optn, contn;

    printf("\nNote: array must be sorted...\n");

    printf("Create an array:\n");

    printf("\nEnter the number of elements (n) in the array: ");

    scanf("%d", &n);

    printf("\nEnter the %d elements of the array:\n", n);

    for(int i = 0; i < n; i++) {

        printf("Element %d: ", i);

        scanf("%d", &arr[i]);

    }

    printf("\nThe array created:\n");

    for(int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    } printf("\n\n");

    do {

        printf("Choose the search you want to perform:\n");

        printf(" 1- Linear search.\n 2- Binary search.\n");

        printf("Enter option: ");

        scanf("%d", &optn);

        switch(optn) {

            case 1:

                printf("\nLinear search:\n");

                printf("Enter the value you want to search: ");

                scanf("%d", &val);

                index = linear\_search(arr, n, val);

                if(index == -1) {

                    printf("Element is not present in given array.\n");

                }

                else {

                    printf("The value %d is present in array at index: %d\n", val, index);

                }

                break;

            case 2:

                printf("\nBinary search:\n");

                printf("Enter the value you want to search: ");

                scanf("%d", &val);

                index = binary\_search(arr, 0, n-1, val);

                if(index == -1) {

                    printf("Element is not present in given array.\n");

                }

                else

                    printf("The value %d is present in array at index: %d\n", val, index);

                break;

            default:

                printf("Incorrect option...\nPlease choose from provided menu.\n");

                break;

        }

        printf("\nDo you want to continue(Y/N): ");

        scanf(" %c", &contn);

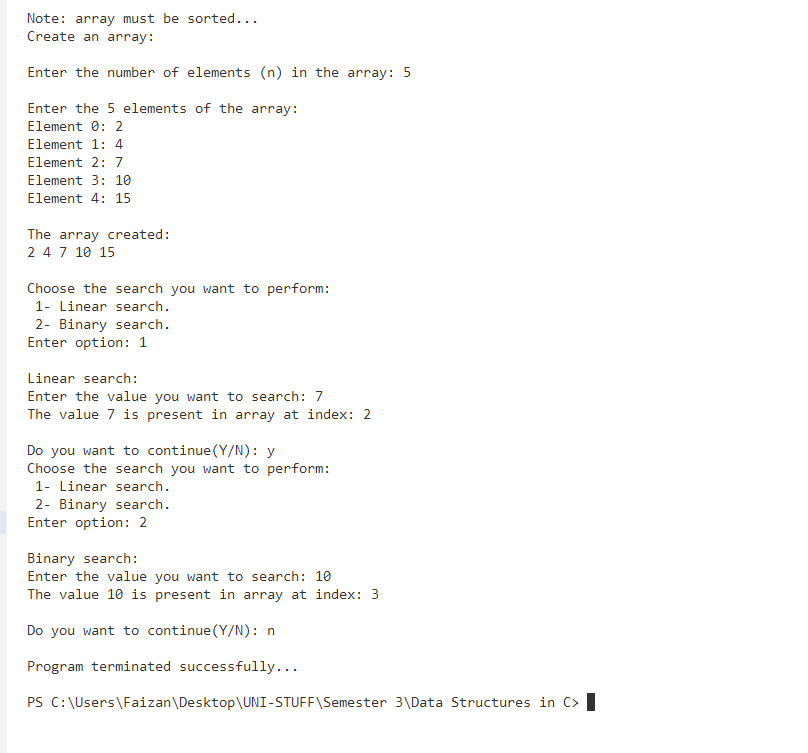
    }

    while(contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n\n");

    return 0;

}

**Output: **

**Lab 4**

**Program:**

Write a menu driven program to perform operations on 2-D arrays. The program must perform the operations: addition, subtraction, multiplication, transpose, diagonal elements sum, print upper triangle, and print lower triangle.

**Source Code:**

#include<stdio.h>

#define MAX 50

void array(int num, int n, int m, int a[MAX][MAX]) {

    printf("Enter the elements of the %d array:\n", num);

    for(int i = 0;i < n; i++) {

        printf("Row %d of array:\n", i);

        for(int j = 0; j < m; j++) {

            printf("Element %d: ", j);

            scanf("%d", &a[i][j]);

        }

    }

    printf("\nArray %d created:\n", num);

    for(int i = 0; i < n; i++) {

        printf("   ");

        for(int j = 0; j < m; j++) {

            printf("%d ", a[i][j]);

        }

        printf("\n");

    } printf("\n");

}

void addition(int n, int m, int a1[MAX][MAX], int a2[MAX][MAX]) {

    int a\_result[MAX][MAX];

    for(int i = 0; i < n; i++) {

        printf(" ");

        for(int j = 0; j < m; j++) {

            a\_result[i][j] = a1[i][j] + a2[i][j];

            printf("%d ", a\_result[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

void subtraction(int n, int m, int a1[MAX][MAX], int a2[MAX][MAX]) {

    int a\_result[MAX][MAX];

    for(int i = 0; i < n; i++) {

        printf(" ");

        for(int j = 0; j < m; j++) {

            a\_result[i][j] = a1[i][j] - a2[i][j];

            printf("%d ", a\_result[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

void multiplication(int n, int m, int a1[MAX][MAX], int a2[MAX][MAX]) {

    int a\_result[MAX][MAX];

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < m; j++) {

            a\_result[i][j] = 0;

            for (int k = 0; k < n; k++) {

                a\_result[i][j] += a1[i][k] \* a2[k][j];

            }

            printf("%d\t", a\_result[i][j]);

        }

        printf("\n");

    }

}

void transpose(int n, int m, int a[MAX][MAX]) {

    int a\_result[MAX][MAX];

    for(int i = 0; i < n; i++) {

        for(int j = 0; j < m; j++) {

            a\_result[i][j] = a[j][i];

            printf("%d ", a\_result[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

void diagonal\_Sum(int n, int m, int a[MAX][MAX], int optn) {

    int sum = 0;

    if(optn == 1) {

        for(int i = 0; i < n ; i++)

            for(int j = 0; j < m; j++) {

                if(i == j) {

                    sum += a[i][j];

                }

            }

    }

    Else {

        for(int i = 0; i < n; i++)

            for(int j = 0; j < m; j++) {

                int c = i + j;

                if(c == (n-1)) {

                    sum += a[i][j];

                }

            }

    }

    printf("Sum of diagonal is: %d\n", sum);

}

void upper\_Triangle(int n, int m, int a[MAX][MAX]) {

    for(int i = 0; i < n; i++) {

        for(int j = 0; j < m; j++) {

            printf("%d ", a[i][j]);

        }

        m--;

        printf("\n");

    }

    printf("\n");

}

void lower\_Triangle(int n, int m, int a[MAX][MAX]) {

    for(int i = 0; i < n; i++) {

        for(int j = 0; j < m; j++) {

            int c = i + j;

            if(c > (n-1)) {

                printf("%d ", a[i][j]);

            }

            else

                printf("  ");

        }

        printf("\n");

    }

    printf("\n");

}

int main() {

    int option, arrChoice;

    int n, m, num, transpose\_of, up\_tri, lo\_tri, optn;

    int a1[MAX][MAX], a2[MAX][MAX];

    char exit;

    //Creating 2 2D arrays from user input:

    printf("Create array 1:\n");

    printf("Enter number of rows of array: ");

    scanf("%d", &n);

    printf("Enter number of columns of array: ");

    scanf("%d", &m);

    array(1, n, m, a1);

    printf("Create array 2:\n");

    array(2, n, m, a2);

    //Printing menu and taking option input:

    printf("\n\*\*\*\*\*\* 2D Array Operations \*\*\*\*\*\*\n\n");

    printf("Choose the operation you want to perform:\n");

    printf(" 0- Create new arrays.\n 1- Addition.\n 2- Subtraction.\n 3- Multiplication.\n 4- Transpose.\n 5- Diagonal sum.\n 6- Upper triangle.\n 7- Lower triangle.");

    do {

        printf("\nSelect operation: ");

        scanf("%d", &option);

        printf("\n");

        switch(option) {

            case 0:

                printf("Create array 1:\n");

                printf("Enter number of rows of array: ");

                scanf("%d", &n);

                printf("Enter number of columns of array: ");

                scanf("%d", &m);

                array(1, n, m, a1);

                printf("Create array 2:\n");

                array(2, n, m, a1);

                break;

            case 1:

                addition(n, m, a1, a2);

                break;

            case 2:

                subtraction(n, m, a1, a2);

                break;

            case 3:

                multiplication(n, m, a1, a2);

                break;

            case 4:

                printf("Choose array to transpose(1/2): ");

                scanf("%d", &transpose\_of);

                if(transpose\_of == 1) {

                    transpose(n, m, a1);

                }

                else {

                    transpose(n, m, a2);

                }

                break;

            case 5:

                printf("Choose diagonal:\n");

                printf(" 1- Decreasing diagonal.\n 2- Increasing diagonal.\n");

                printf("Enter option: ");

                scanf("%d", &optn);

                printf("Choose array to sum the diagonal of(1/2): ");

                scanf("%d", &arrChoice);

                if(arrChoice == 1) {

                    diagonal\_Sum(n, m, a1, optn);

                }

                else {

                    diagonal\_Sum(n, m, a2, optn);

                }

                break;

            case 6:

                printf("Choose array to print upper triangle of(1/2): ");

                scanf("%d", &up\_tri);

                if(up\_tri == 1) {

                    upper\_Triangle(n, m, a1);

                }

                else {

                    upper\_Triangle(n, m, a2);

                }

                break;

            case 7:

                printf("Choose array to print lower triangle of(1/2): ");

                scanf("%d", &lo\_tri);

                if(lo\_tri == 1) {

                    lower\_Triangle(n, m, a1);

                }

                else {

                    lower\_Triangle(n, m, a2);

                }

                break;

            default:

                printf("Wrong option...\n");

                printf("Choose from given options (1 - 7).\n\n");

                break;

        }

        printf("Do you want to continue (Y/N): ");

        scanf(" %c", &exit);

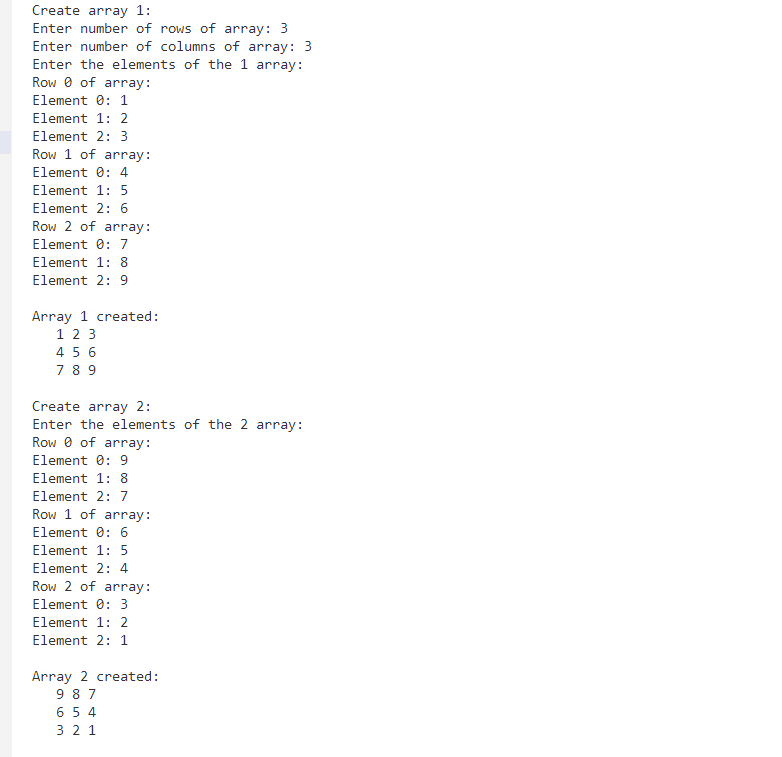
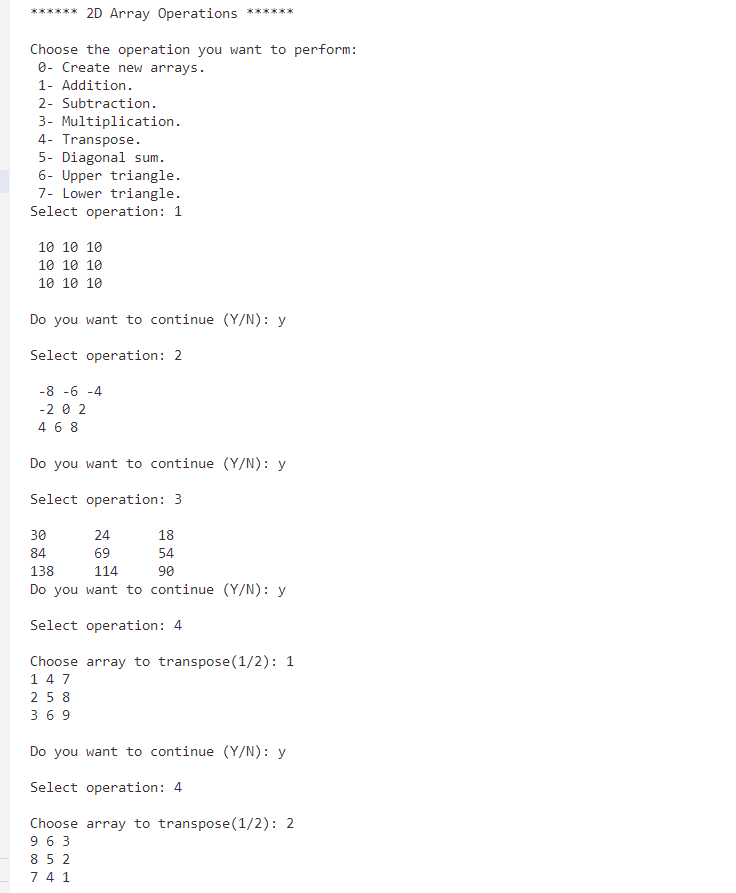
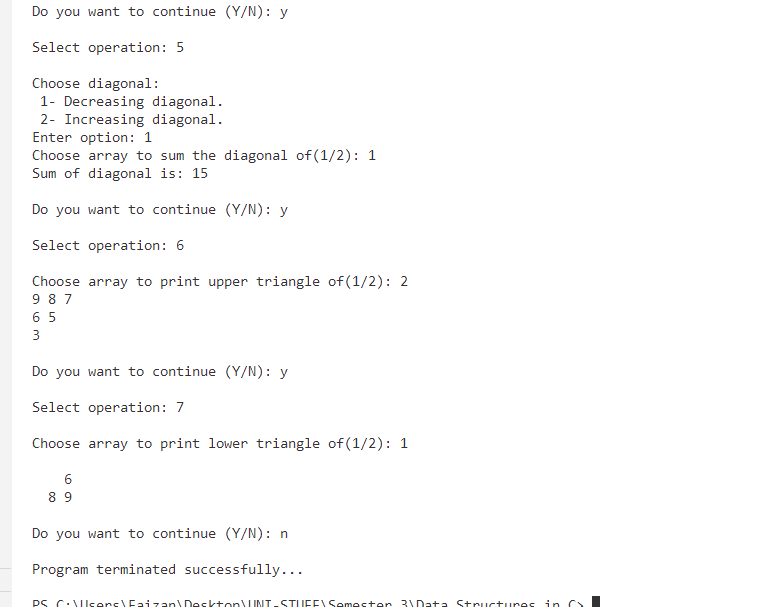
    }

    while(exit == 'Y' || exit == 'y');

    printf("\nProgram terminated successfully...\n\n");

    return 0;

}

**Output: ** **** 

**Lab 5**

**Program:**

Write a program to pass an array as a parameter to a function.

**Source Code:**

#include<stdio.h>

#define MAX 50

int array\_multiple(int n, int m, int x, int arr[MAX][MAX]) {

    int new\_arr[MAX][MAX];

    for(int i = 0; i < n; i++) {

        for(int j = 0; j < m; j++) {

            new\_arr[i][j] = arr[i][j] \* x;

            printf("%d ", new\_arr[i][j]); }

        printf("\n");

    }

}

int main() {

    int n, m, x;

    int arr[MAX][MAX];

    printf("Create a 2-D array:\n");

    printf("Enter number of rows: ");

    scanf("%d", &n);

    printf("Enter number of columns: ");

    scanf("%d", &m);

    printf("Enter the elements of the array:\n");

    for(int i = 0;i < n; i++) {

        printf("Row %d of array:\n", i);

        for(int j = 0; j < m; j++) {

            printf("Element %d: ", j);

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

    }

    printf("\nArray created:\n");

    for(int i = 0; i < n; i++) {

        printf("   ");

        for(int j = 0; j < m; j++){

            printf("%d ", arr[i][j]);

        }

        printf("\n");

    } printf("\n");

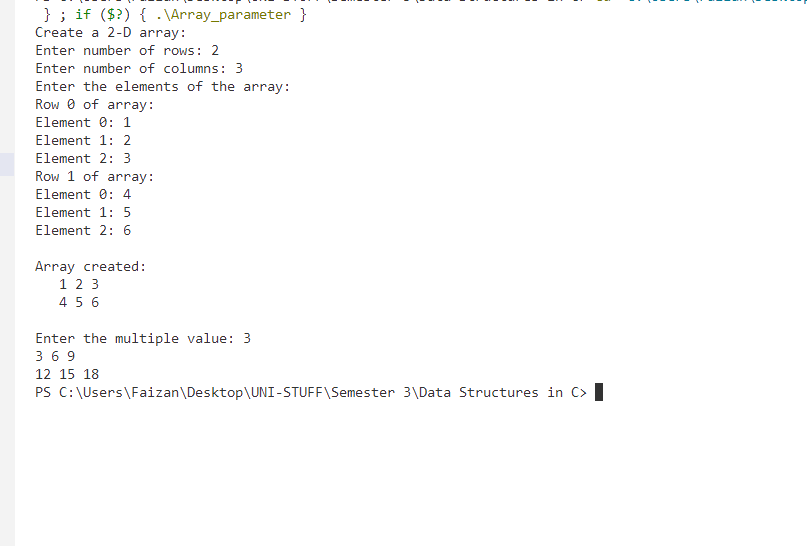
    printf("Enter the multiple value: ");

    scanf("%d", &x);

    array\_multiple(n, m, x, arr);

    return 0;

}

**Output: **

**Lab 6**

**Program:**

Write a menu driven program to implement the stack operations of PUSH, POP, and Display.

**Source Code:**

#include<stdio.h>

#define MAX 5

int stack[MAX];

int top = -1;

void PUSH(int val) {

    printf("\nEnter value to push to stack: ");

    scanf("%d", &val);

    if(top == (MAX - 1)) {

        printf("\nStack is full.\nCannot PUSH new value to stack.\n");

    }

    else {

        top++;

        stack[top] = val;

        printf("%d pushed to stack.\n", val);

    }

}

void POP() {

    if(top == -1) {

        printf("Stack is empty.\n");

    }

    else {

        printf("%d popped from stack.\n", stack[top]);

        top--;

    }

}

void display() {

    if(top == -1) {

        printf("Stack is empty.\nNo Display...\n");

    }

    else {

        printf("Stack Display:\n");

        for(int i = top; i >= 0; i--) {

            printf("    %d\n", stack[i]);

        }

    }

}

int main() {

    int optn, val;

    char contn;

    printf("Stack operations:\n 1- PUSH.\n 2- POP.\n 3- Display stack.\n");

    do {

        printf("Enter option (1-3): ");

        scanf("%d", &optn);

        switch(optn) {

            case 1:

                PUSH(val);

                break;

            case 2:

                POP();

                break;

            case 3:

                display();

                break;

            default:

                printf("Incorrect option.\nPlease choose from given options.\n");

                break;

        }

        printf("\nDo you want to continue (Y/N): ");

        scanf(" %c", &contn);

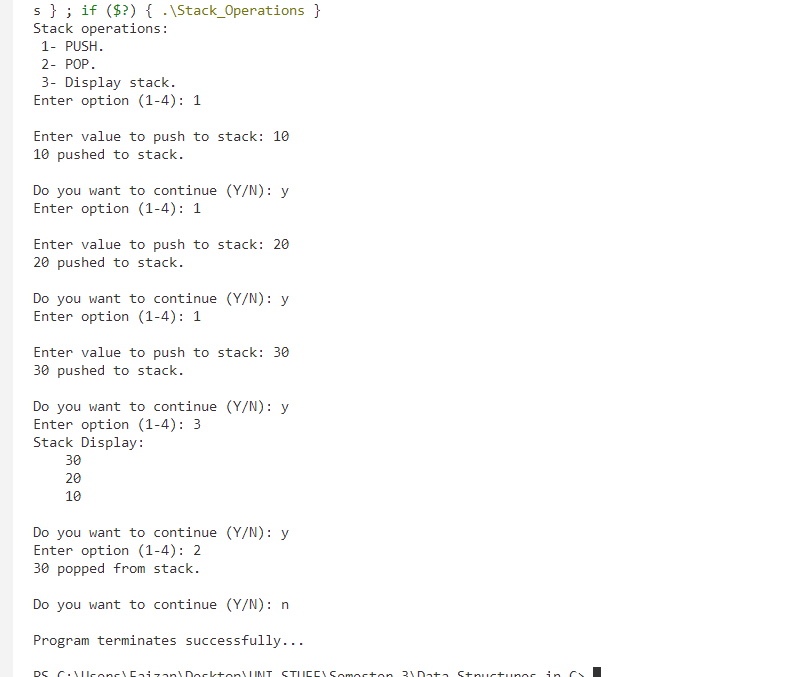
    }

    while(contn == 'Y' || contn == 'y');

    printf("\nProgram terminates successfully...\n\n");

    return 0;

}

**Output: **

**Lab 7**

**Program:**

Write a menu driven program to implement the queue operations of enqueue, dequeue, and display.

**Source Code:**

#include<stdio.h>

#define MAX 10

int Q[MAX];

int F = -1, R = -1;

void enqueue(int val) {

    if(R == (MAX - 1)) {

        printf("Queue is full.\n");

    }

    else if(F == -1 && R == -1) {

        F = 0;

        R = 0;

        printf("\nEnter value to add to queue: ");

        scanf("%d", &val);

        Q[R] = val;

        printf("%d added to queue.\n", val);

    }

    else {

        R++;

        printf("\nEnter value to add to queue: ");

        scanf("%d", &val);

        Q[R] = val;

        printf("%d added to queue.\n", val);

    }

}

void dequeue() {

    if(F == -1 && R == -1) {

        printf("Queue is already empty.\n");

    }

    else if(F == (MAX - 1) && R == (MAX - 1)) {

        F = -1;

        R = -1;

    }

    else {

        printf("%d deleted from queue.\n", Q[F]);

        F++;

    }

}

void display() {

    printf("\nDisplay Queue:\n");

    printf("  ");

    for(int i = F; i <= R; i++)

        printf("%d ", Q[i]);

    printf("\n");

}

int main() {

    int val, optn;

    char contn;

    printf("\nQueue operations:\n 1- Insert in queue.\n 2- Delete from queue.\n 3- Display queue.\n");

    do {

        printf("Enter option (1-3): ");

        scanf("%d", &optn);

        switch(optn) {

            case 1:

                enqueue(val);

                break;

            case 2:

                dequeue();

                break;

            case 3:

                display();

                break;

            default:

                printf("Incorrect option.\nChoose from given options (1-3).\n");

                break;

        }

        printf("\nDo you want to continue (Y/N): ");

        scanf(" %c", &contn);

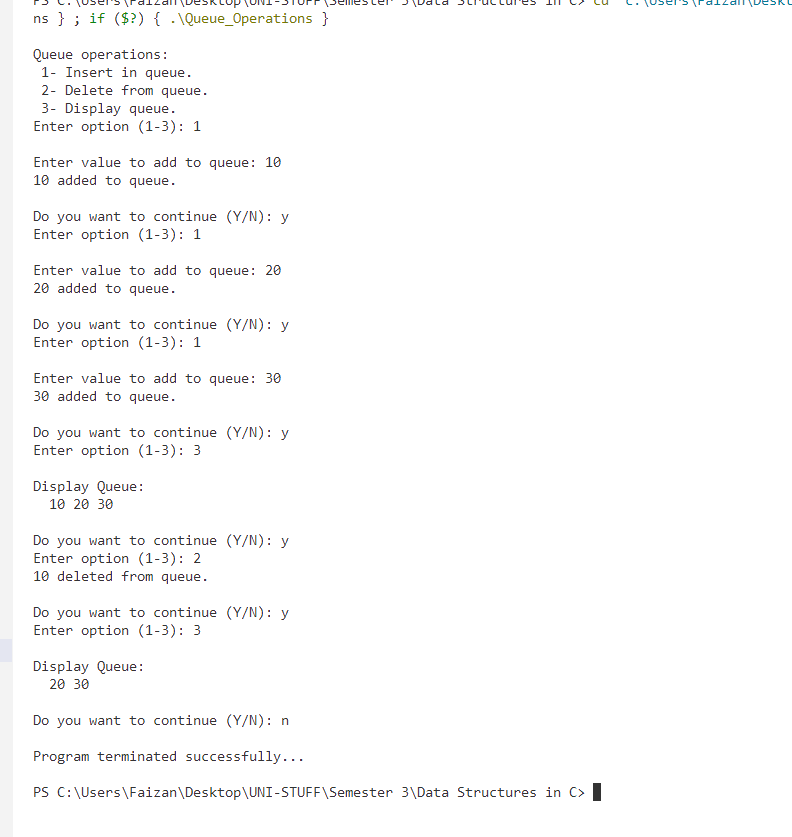
    }

    while(contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n\n");

    return 0;

}

**Output:** ****

**Lab 8**

**Program:**

Write a program to take a string of parenthesis as input and to check whether the parenthesis are balanced or not with the use of stacks.

**Source Code:**

#include<stdio.h>

#include<string.h>

#define MAX 100

void parenthesis\_check(char s[]) {

    char stack[MAX];

    int stack\_index = -1;

    int len = strlen(s);

    char input[MAX];

    int input\_index = -1;

    int flag = 0;

    for(int i = 0; i < len; i++) {

        char c = s[i];

        if(c == '(' || c == '[' || c == '{') {

            stack\_index++;

            stack[stack\_index]  = c;

        }

        else if(c == ')' || c == ']' || c == '}') {

            if(stack\_index > -1) {

                if((c == ')' && stack[stack\_index] == '(') || (c == ']' && stack[stack\_index] == '[') || (c == '}' && stack[stack\_index] == '{')) {

                    stack\_index--;

                    // printf("%d \n", stack\_index);

                }

                else {

                    // printf("Unbalanced!\n");

                    flag = 1;

    break;

                }

            }

            else if(stack\_index == -1) {

                // printf("Unbalanced!\n");

                flag = 1;

    break;

            }

        }

    }

    if(stack\_index == -1) {

        if(flag == 0) {

            printf("\nParenthesis string is balanced...\n\n");

        }

        else {

            printf("\nStirng is unbalanced...\n\n");

        }

    }

    else if(stack\_index >= 0) {

        printf("\nString is unbalanced...\n\n");

    }

}

int main() {

    char string[MAX];

    printf("Parenthesis Checker:\n\n");

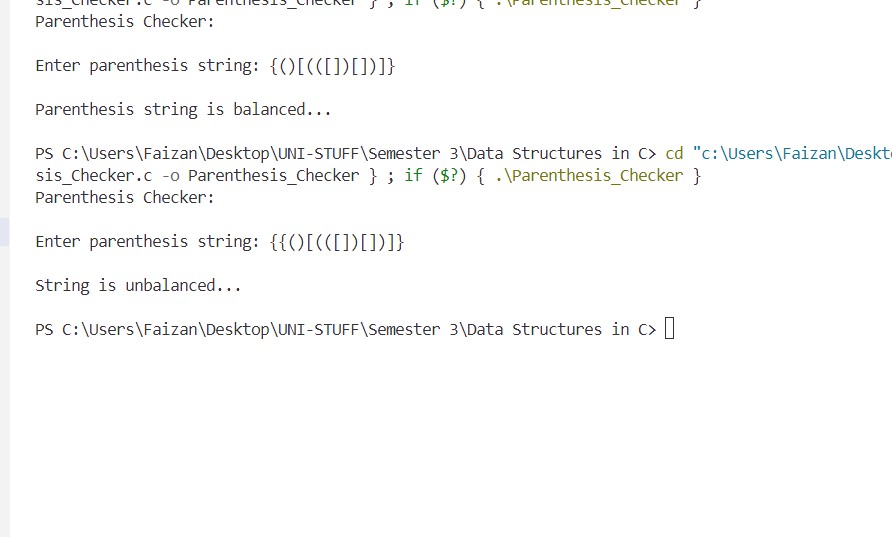
    printf("Enter parenthesis string: ");

    fgets(string, MAX, stdin);

    parenthesis\_check(string);

    return 0;

}

**Output:**

**Lab 9**

**Program:**

Write a program to perform infix to prefix conversion on an expression and to perform evaluation of the prefix expression.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

int precedence(char c) {

    if (c == '^')

        return 3;

    else if (c == '/' || c == '\*')

        return 2;

    else if (c == '+' || c == '-')

        return 1;

    else

        return -1;

}

void ReverseString(char\* str) {

    int len = strlen(str);

    for (int i = 0; i < len / 2; i++) {

        char temp = str[i];

        str[i] = str[len - 1 - i];

        str[len - 1 - i] = temp;

    }

}

void Prefix(char s[]) {

    char exp[MAX];

    int len = strlen(s);

    char stack[MAX];

    int stack\_index = -1;

    int output\_index = 0;

    ReverseString(s);

    for (int i = 0; i < len; i++) {

        char c = s[i];

        if (isdigit(c) || isalpha(c)) {

            exp[output\_index++] = c;

        } else if (c == ')') {

            stack[++stack\_index] = c;

        } else if (c == '(') {

            while (stack\_index >= 0 && stack[stack\_index] != ')') {

                exp[output\_index++] = stack[stack\_index--];

            }

            stack\_index--;

        } else {

            while (stack\_index >= 0 && precedence(c) < precedence(stack[stack\_index])) {

                exp[output\_index++] = stack[stack\_index--];

            }

            stack[++stack\_index] = c;

        }

    }

    while (stack\_index >= 0) {

        exp[output\_index++] = stack[stack\_index--];

    }

    exp[output\_index] = '\0';

    ReverseString(exp);

    printf("Prefix expression:\n %s\n", exp);

    int evaluation[MAX];

    int eval\_index = -1;

    for (int i = strlen(exp) - 1; i >= 0; i--) {

        char e = exp[i];

        if (isdigit(e)) {

            evaluation[++eval\_index] = e - '0';

        } else if (e == '+' || e == '-' || e == '\*' || e == '/') {

            int a = evaluation[eval\_index--];

            int b = evaluation[eval\_index--];

            int result = 0;

            if (e == '+') {

                result = a + b;

                printf("%d + %d = %d\n", a, b, result);

            } else if (e == '-') {

                result = a - b;

                printf("%d - %d = %d\n", a, b, result);

            } else if (e == '\*') {

                result = a \* b;

                printf("%d \* %d = %d\n", a, b, result);

            } else if (e == '/') {

                result = a / b;

                printf("%d / %d = %d\n", a, b, result);

            }

            evaluation[++eval\_index] = result;

        }

    }

    printf("Evaluation of prefix expression:\n %d\n", evaluation[eval\_index]);

}

int main() {

    char exp[MAX];

    char contn;

    do {

        printf("Enter expression: ");

        fgets(exp, MAX, stdin);

        exp[strcspn(exp, "\n")] = '\0';

        Prefix(exp);

        int ch;

        while ((ch = getchar()) != '\n' && ch != EOF);

        printf("Do you want to continue (Y/N): ");

        scanf(" %c", &contn);

        while ((ch = getchar()) != '\n' && ch != EOF);

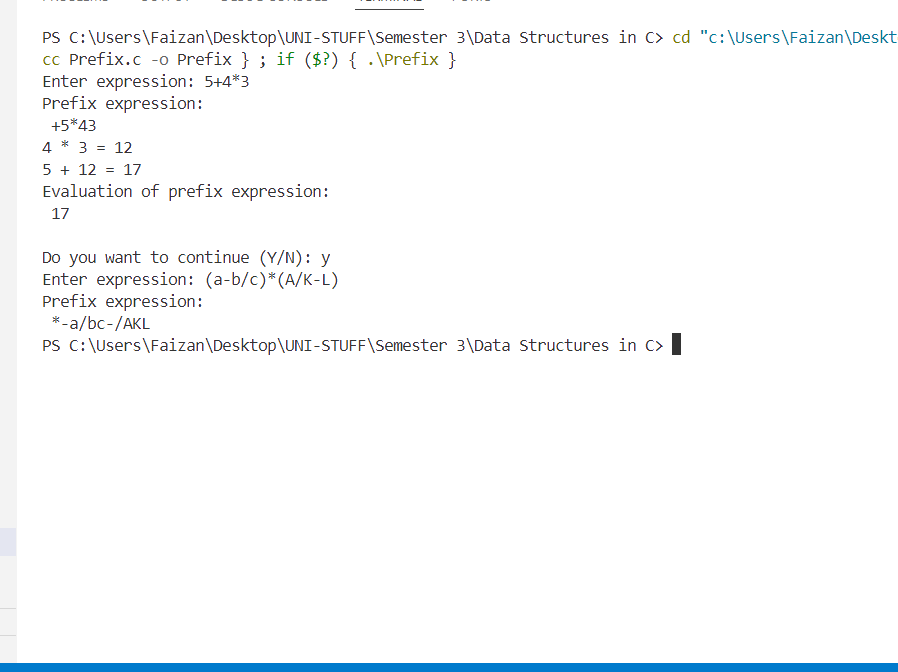
    }

    while(contn == 'y' || contn == 'Y');

    printf("\nProgram terminated successfully...\n\n");

    return 0;

}

**Output:** ****

**Lab 10**

**Program:**

Write a program to perform infix to postfix conversion on an expression and to perform evaluation of the postfix expression.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

int precedence(char c) {

    if (c == '^')

        return 3;

    else if (c == '/' || c == '\*')

        return 2;

    else if (c == '+' || c == '-')

        return 1;

    else

        return -1;

}

void Postfix(char s[]) {

    char output\_exp[MAX];

    int output\_index = 0;

    int len = strlen(s);

    char stack[MAX];

    int stack\_index = -1;

    for (int i = 0; i < len; i++) {

        char c = s[i];

        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {

            output\_exp[output\_index++] = c;

        }

        else if (c == '(') {

            stack[++stack\_index] = c;

        }

        else if (c == ')') {

            while (stack\_index >= 0 && stack[stack\_index] != '(') {

                output\_exp[output\_index++] = stack[stack\_index--];

            }

            stack\_index--;

        }

        else {

            while (stack\_index >= 0 && (precedence(s[i]) < precedence(stack[stack\_index]) ||

                                       precedence(s[i]) == precedence(stack[stack\_index]))) {

                output\_exp[output\_index++] = stack[stack\_index--];

            }

            output\_exp[output\_index++] = ' ';

            stack[++stack\_index] = c;

        }

    }

    while (stack\_index >= 0) {

        output\_exp[output\_index++] = stack[stack\_index--];

        output\_exp[output\_index++] = ' ';

    }

    output\_exp[output\_index] = '\0';

    printf("Postfix expression:\n %s\n", output\_exp);

    int evaluation[MAX];

    int eval\_index = -1;

    int i = 0;

    while (output\_exp[i] != '\0') {

        if (isdigit(output\_exp[i])) {

            int num = 0;

            while (isdigit(output\_exp[i])) {

                num = num \* 10 + (output\_exp[i] - '0');

                i++;

            }

            evaluation[++eval\_index] = num;

        } else if (output\_exp[i] == ' ') {

            i++;

        } else if (output\_exp[i] == '+' || output\_exp[i] == '-' || output\_exp[i] == '\*' || output\_exp[i] == '/') {

            int b = evaluation[eval\_index--];

            int a = evaluation[eval\_index--];

            int result = 0;

            if (output\_exp[i] == '+') {

                result = a + b;

                printf("%d + %d = %d\n", a, b, result);

            } else if (output\_exp[i] == '-') {

                result = a - b;

                printf("%d - %d = %d\n", a, b, result);

            } else if (output\_exp[i] == '\*') {

                result = a \* b;

                printf("%d \* %d = %d\n", a, b, result);

            } else if (output\_exp[i] == '/') {

                result = a / b;

                printf("%d / %d = %d\n", a, b, result);

            }

            evaluation[++eval\_index] = result;

            i++;

        } else {

            i++;

        }

    }

    printf("Evaluation of expression:\n %d\n", evaluation[eval\_index]);

}

int main() {

    char exp[MAX];

    char contn;

    int ch;

    do {

        printf("Enter expression: ");

        fgets(exp, MAX, stdin);

        exp[strcspn(exp, "\n")] = '\0';

        Postfix(exp);

        while ((ch = getchar()) != '\n' && ch != EOF);

        printf("Do you want to continue (Y/N): ");

        scanf(" %c", &contn);

        while ((ch = getchar()) != '\n' && ch != EOF);

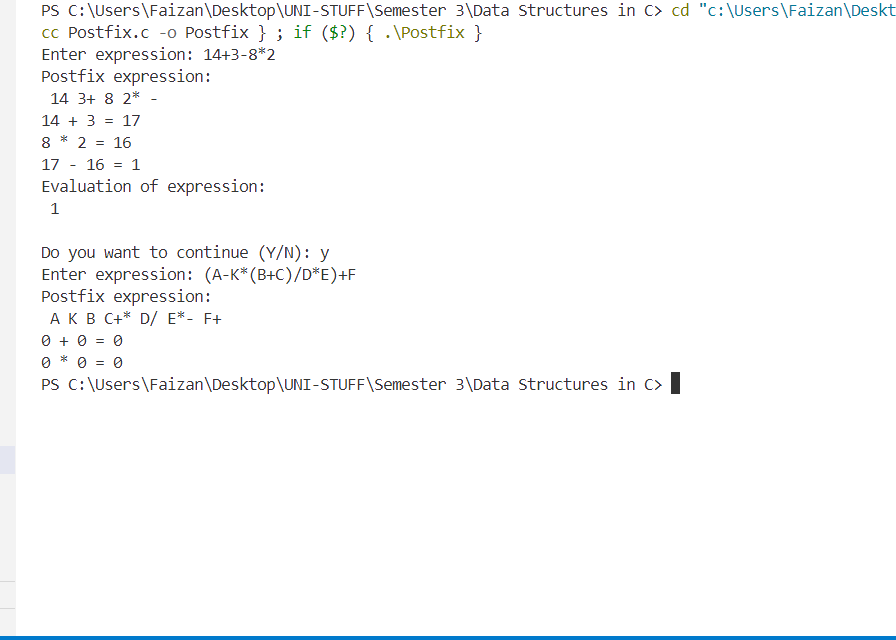
    }

    while (contn == 'y' || contn == 'Y');

    printf("\nProgram terminated successfully...\n\n");

    return 0;

}

**Output:** ****

**Lab 11**

**Program:**

Write a menu driven program to perform the operations of Tower of Hanoi, Factorial, and Fibonacci.

**Source code:**

#include<stdio.h>

int step = 1;

void TOH(int n, char S, char D, char A){

    if(n == 1) {

        printf("Step %d: Move disk from tower %d to tower %d\n", step++, S, D);

    }

    else{

        TOH(n-1, S, A, D);

        printf("Step %d: Move disk from tower %d to tower %d.\n", step++, S, D);

        TOH(n-1, A, D, S);

    }

}

int Factorial(int n){

    if(n == 0 || n == 1)

        return 1;

    else

        return n \* Factorial(n-1);

}

void Fibonacci(int n){

    int a = 0, b = 1, next;

    if(n == 0) {

        printf("Enter a positive number.\n");

        return;

    }

    printf("Fibonacci for %d terms:\n", n);

    for(int i = 0; i < n; i++) {

        if(i == 0)

            printf("%d ", a);

        else if(i == 1)

            printf("%d ", b);

        else {

            next = a + b;

            a = b;

            b = next;

            printf("%d ", next);

        }

    }

    printf("\n");

}

int main(){

    int n, optn;

    int Tsource, Tdest, Taux;

    char contn;

    printf("Operations:\n 1- Tower of Hanoi.\n 2- Factorial.\n 3- Fibonacci.\n");

    do {

        printf("Enter option: ");

        scanf("%d", &optn);

        switch(optn) {

            case 1:

                printf("Enter number of rings: ");

                scanf("%d", &n);

                printf("Enter source tower: ");

                scanf("%d", &Tsource);

                printf("Enter destination tower: ");

                scanf("%d", &Tdest);

                TOH(n, Tsource, Tdest, Taux);

                printf("Total steps: %d\n", step-1);

                break;

            case 2:

                printf("Enter number for factorial: ");

                scanf("%d", &n);

                printf("Factorial of %d is: %d", n, Factorial(n));

                break;

            case 3:

                printf("Enter number of terms in Fibonacci: ");

                scanf("%d", &n);

                Fibonacci(n);

                break;

            default:

                printf("\nIncorrect option.\nChoose from given options(1-3).\n");

                break;

        }

        printf("\nDo you want to continue(Y/N): ");

        scanf(" %c", &contn);

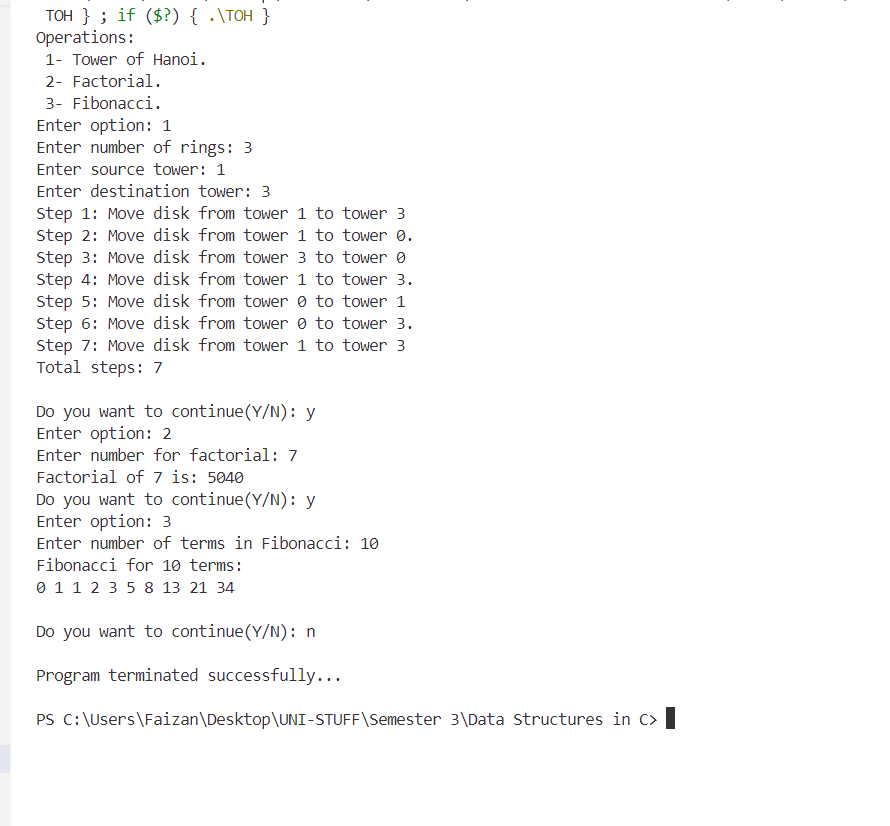
    }

    while(contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n\n");

    return 0;

}

**Output:** ****

**Lab 12.1**

**Program:**

Write a program to perform Bubble sorting on an array of size n. Take the elements and size of array from user.

**Source code:**

#include<stdio.h>

#define MAX 100

void bubble\_sort(int n, int a[MAX]){

    for(int i = 0; i < n; i++){

        for(int j = 0; j < (n-1); j++){

            if(a[j] > a[j+1]) {

                int temp = a[j];

                a[j] = a[j+1];

                a[j+1] = temp;

            }

        }

    }

    printf("Sorted array:\n");

    for(int i = 0; i < n; i++){

        printf("%d ", a[i]);

    }

}

int main(){

    int n, arr[MAX];

    printf("Enter number of elements in array: ");

    scanf("%d", &n);

    for(int i = 0; i < n; i++){

        printf("Element %d: ", i);

        scanf("%d", &arr[i]);

    }

    printf("Array created:\n");

    for(int i = 0; i < n; i++)

    {

        printf("%d ", arr[i]);

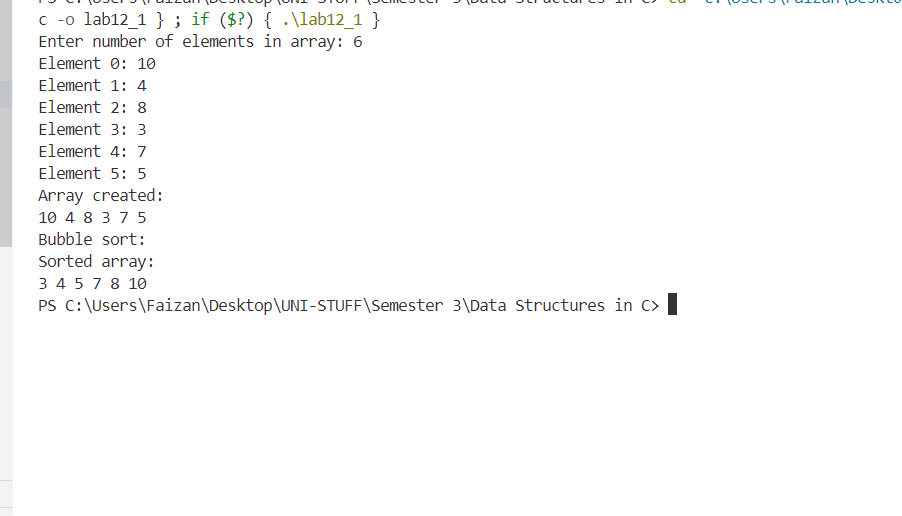
    }

    printf("\nBubble sort:\n");

    bubble\_sort(n, arr);

    return 0;

}

**Output:** ****

**Lab 12.2**

**Program:**

Write a program to perform Selection sorting on an array of size n. Take the elements and size of array from user.

**Source code:**

#include <stdio.h>

#define MAX 100

void selectionSort(int arr[MAX], int n) {

    for (int i = 0; i < n - 1; i++) {

        int min = i;

        for (int j = i + 1; j < n; j++) {

            if (arr[j] < arr[min]) {

                min = j;

            }

        }

        int temp = arr[min];

        arr[min] = arr[i];

        arr[i] = temp;

    }

}

int main() {

    int arr[MAX];

    int n;

    printf("Enter n: ");

    scanf("%d", &n);

    printf("Create array:\n");

    for(int i = 0; i < n; i++){

        printf("Element %d: ", i);

        scanf("%d", &arr[i]);

    }

    printf("Unsorted array: \n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

    selectionSort(arr, n);

    printf("Sorted array: \n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

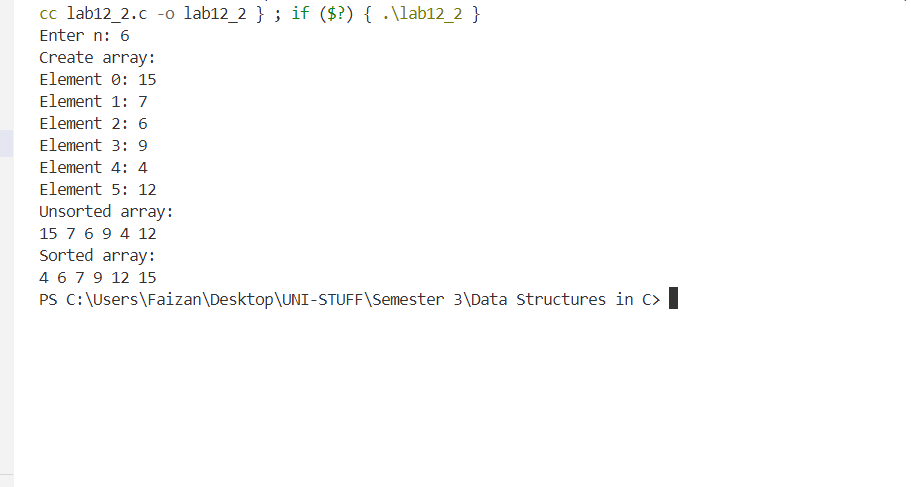
    }

    printf("\n");

    return 0;

}

**Output:**

****

**Lab 12.3**

**Program:**

Write a program to perform Insertion sorting on an array of size n. Take the elements and size of array from user.

**Source code:**

#include<stdio.h>

#define MAX 100

void insertionSort(int n, int arr[MAX]){

    for(int i = 0; i < n; i++){

        int key = arr[i];

        int j = i-1;

        while(j >= 0 && arr[j] > key){

            arr[j+1] = arr[j];

            j = j - 1;

        }

        arr[j+1] = key;

    }

}

int main(){

    int n, arr[MAX];

    printf("Enter n: ");

    scanf("%d", &n);

    printf("Create array:\n");

    for(int i = 0; i < n; i++){

        printf("Element %d: ", i);

        scanf("%d", &arr[i]);

    }

    printf("Unsorted array: \n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("Insertion sort:\n");

    insertionSort(n, arr);

    printf("Sorted array: \n");

    for (int i = 0; i < n; i++) {

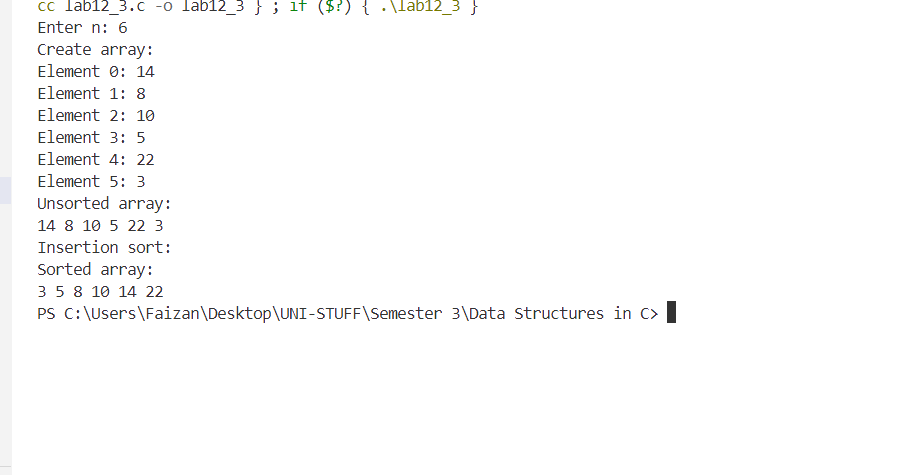
        printf("%d ", arr[i]);

    }

    printf("\n");

    return 0;

}

**Output:** ****

**Lab 13.1**

**Program:**

Write a program to perform Insertion and Deletion from the beginning, middle, and the end of a Singly Linked List.

**Source code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct Node {

    int data;

    struct Node\* next;

} Node;

Node\* createNode(int data) {

    Node\* NN = (Node\*)malloc(sizeof(Node));

    if(NN == NULL) {

        printf("List overflow...\n");

        exit(1);

    }

    NN->data = data;

    NN->next = NULL;

    return NN;

}

void insert\_beginning(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    NN->next = \*start;

    \*start = NN;

}

void insert\_middle(Node\*\* start, int data, int position) {

    if(position < 0){

        printf("Invalid position...\n");

        return;

    }

    if(position == 0){

        insert\_beginning(start, data);

        return;

    }

    Node\* NN = createNode(data);

    Node\* temp = \*start;

    for(int i = 0; temp != NULL && i < (position-1); i++)

    {

        temp = temp->next;

    }

    if(temp == NULL) {

        printf("Incorrect position...\n");

        free(NN);

        return;

    }

    NN->next = temp->next;

    temp->next = NN;

}

void insert\_end(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    if(\*start == NULL) {

        \*start = NN;

        return;

    }

    Node\* end = \*start;

    while(end->next != NULL) {

        end = end->next;

    }

    end->next = NN;

}

void delete\_beginning(Node\*\* start) {

    if(\*start == NULL){

        printf("List underflow.\n");

        return;

    }

    Node\* temp = \*start;

    \*start = temp->next;

    free(temp);

}

void delete\_middle(Node\*\* start, int position) {

    if(\*start == NULL){

        printf("List underflow.\n");

        return;

    }

    if(position == 0){

        delete\_beginning(start);

        return;

    }

    Node\* temp = \*start;

    for (int i = 0; temp != NULL && i < position - 1; i++) {

        temp = temp->next;

    }

    if (temp == NULL || temp->next == NULL) {

        printf("Incorrect position.\n");

        return;

    }

    Node\* NN = temp->next;

    temp->next = NN->next;

    free(NN);

}

void delete\_end(Node\*\* start) {

    if(\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    if((\*start)->next == NULL) {

        free(\*start);

        \*start = NULL;

        return;

    }

    Node\* temp = \*start;

    while((temp->next)->next != NULL) {

        temp = temp->next;

    }

    free(temp->next);

    temp->next = NULL;

}

void display(Node\* node) {

    while (node != NULL) {

        printf("%d -> ", node->data);

        node = node->next;

    }

    printf("NULL\n");

}

int main()

{

    Node\* start = NULL;

    int val, position, optn;

    char contn;

    printf("Linked List:\n Insert:\n  1- Beginning.\n  2- Middle.\n  3- End.\n Delete:\n  4- Beginning.\n  5- Middle.\n  6- End.\n  7- Display.\n\n");

    do {

        printf("Enter option: ");

        scanf("%d", &optn);

        switch(optn){

            case 1:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_beginning(&start, val);

                break;

            case 2:

                printf("Enter value: ");

                scanf("%d", &val);

                printf("Enter position: ");

                scanf("%d", &position);

                insert\_middle(&start, val, position);

                break;

            case 3:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_end(&start, val);

                break;

            case 4:

                delete\_beginning(&start);

                break;

            case 5:

                printf("Enter position: ");

                scanf("%d", &position);

                delete\_middle(&start, position);

                break;

            case 6:

                delete\_end(&start);

                break;

            case 7:

                printf("Linked List display:\n");

                display(start);

                break;

            default:

                printf("Incorrect option...\n");

                break;

        }

        printf("Do you want to continue(Y/N): ");

        scanf(" %c", &contn);

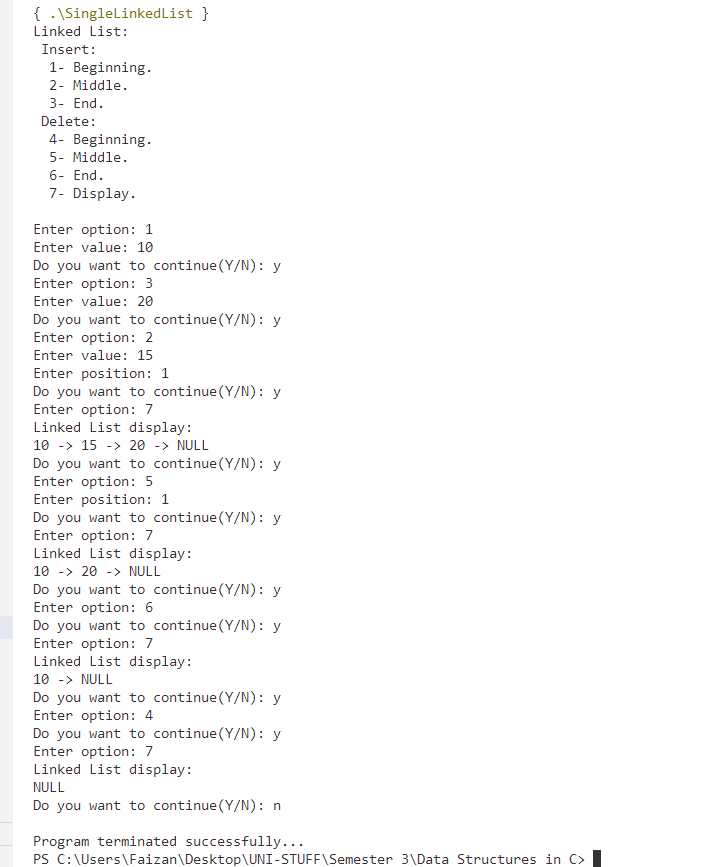
    }

    while(contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n");

    return 0;

}

**Output:** ****

**Lab 13.2**

**Program:**

Write a program to perform Insertion and Deletion from the beginning, middle, and the end of a Doubly Linked List.

**Source code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

    int data;

    struct Node\* next;

    struct Node\* prev;

} Node;

Node\* createNode(int data) {

    Node\* NN = (Node\*)malloc(sizeof(Node));

    if (NN == NULL) {

        printf("List overflow...\n");

        exit(1);

    }

    NN->data = data;

    NN->next = NULL;

    NN->prev = NULL;

    return NN;

}

void insert\_beginning(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    NN->next = \*start;

    if (\*start != NULL) {

        (\*start)->prev = NN;

    }

    \*start = NN;

}

void insert\_middle(Node\*\* start, int data, int position) {

    if (position < 0) {

        printf("Invalid position...\n");

        return;

    }

    if (position == 0) {

        insert\_beginning(start, data);

        return;

    }

    Node\* NN = createNode(data);

    Node\* temp = \*start;

    for (int i = 0; temp != NULL && i < (position - 1); i++) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Incorrect position...\n");

        free(NN);

        return;

    }

    NN->next = temp->next;

    NN->prev = temp;

    if (temp->next != NULL) {

        temp->next->prev = NN;

    }

    temp->next = NN;

}

void insert\_end(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    if (\*start == NULL) {

        \*start = NN;

        return;

    }

    Node\* end = \*start;

    while (end->next != NULL) {

        end = end->next;

    }

    end->next = NN;

    NN->prev = end;

}

void delete\_beginning(Node\*\* start) {

    if (\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    Node\* temp = \*start;

    \*start = temp->next;

    if (\*start != NULL) {

        (\*start)->prev = NULL;

    }

    free(temp);

}

void delete\_middle(Node\*\* start, int position) {

    if (\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    if (position == 0) {

        delete\_beginning(start);

        return;

    }

    Node\* temp = \*start;

    for (int i = 0; temp != NULL && i < position; i++) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Incorrect position.\n");

        return;

    }

    if (temp->prev != NULL) {

        temp->prev->next = temp->next;

    }

    if (temp->next != NULL) {

        temp->next->prev = temp->prev;

    }

    free(temp);

}

void delete\_end(Node\*\* start) {

    if (\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    if ((\*start)->next == NULL) {

        free(\*start);

        \*start = NULL;

        return;

    }

    Node\* temp = \*start;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->prev->next = NULL;

    free(temp);

}

void display(Node\* node) {

    while (node != NULL) {

        printf("%d <-> ", node->data);

        node = node->next;

    }

    printf("NULL\n");

}

int main() {

    Node\* start = NULL;

    int val, position, optn;

    char contn;

    printf("Doubly Linked List:\n Insert:\n  1- Beginning.\n  2- Middle.\n  3- End.\n Delete:\n  4- Beginning.\n  5- Middle.\n  6- End.\n  7- Display.\n\n");

    do {

        printf("Enter option: ");

        scanf("%d", &optn);

        switch (optn) {

            case 1:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_beginning(&start, val);

                break;

            case 2:

                printf("Enter value: ");

                scanf("%d", &val);

                printf("Enter position: ");

                scanf("%d", &position);

                insert\_middle(&start, val, position);

                break;

            case 3:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_end(&start, val);

                break;

            case 4:

                delete\_beginning(&start);

                break;

            case 5:

                printf("Enter position: ");

                scanf("%d", &position);

                delete\_middle(&start, position);

                break;

            case 6:

                delete\_end(&start);

                break;

            case 7:

                printf("Doubly Linked List display:\n");

                display(start);

                break;

            default:

                printf("Incorrect option...\n");

                break;

        }

        printf("Do you want to continue(Y/N): ");

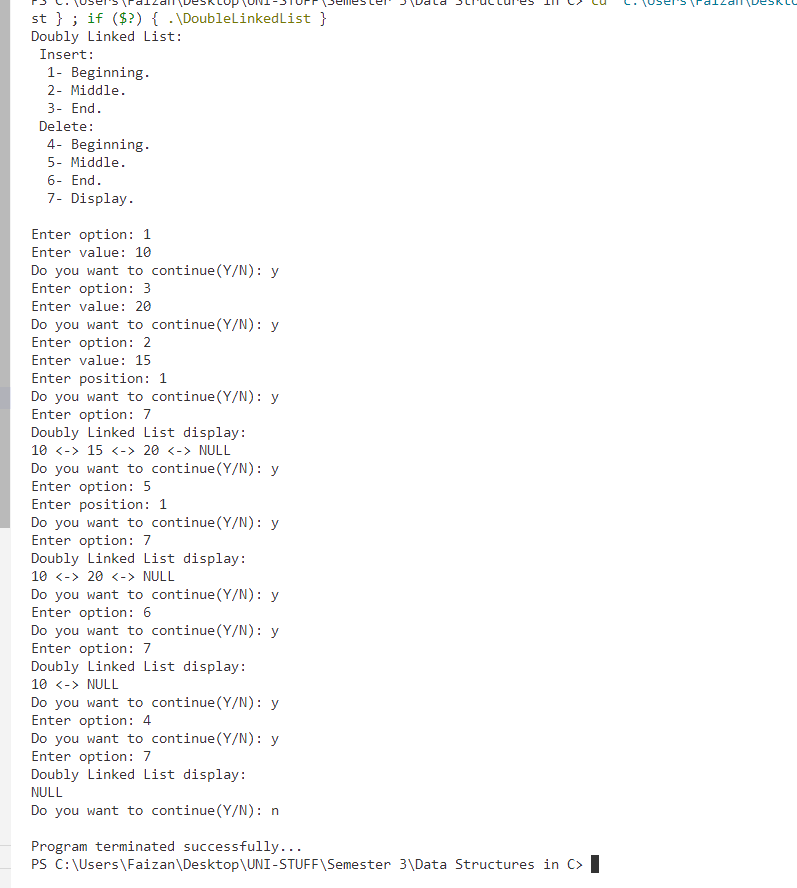
        scanf(" %c", &contn);

    } while (contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n");

    return 0;

}

**Output:** ****

**Lab 14.1**

**Program:**

Write a program to perform Insertion and Deletion from the beginning, middle, and the end of a circular Singly Linked List.

**Source code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct Node {

    int data;

    struct Node\* next;

} Node;

Node\* createNode(int data) {

    Node\* NN = (Node\*)malloc(sizeof(Node));

    if(NN == NULL) {

        printf("List overflow...\n");

        exit(1);

    }

    NN->data = data;

    NN->next = NULL;

    return NN;

}

void insert\_beginning(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    if(\*start == NULL) {

        \*start = NN;

        NN->next = \*start;

    } else {

        Node\* temp = \*start;

        while(temp->next != \*start) {

            temp = temp->next;

        }

        temp->next = NN;

        NN->next = \*start;

        \*start = NN;

    }

}

void insert\_middle(Node\*\* start, int data, int position) {

    if(position < 0) {

        printf("Invalid position...\n");

        return;

    }

    if(position == 0) {

        insert\_beginning(start, data);

        return;

    }

    Node\* NN = createNode(data);

    Node\* temp = \*start;

    for(int i = 0; i < position - 1 && temp->next != \*start; i++) {

        temp = temp->next;

    }

    if(temp->next == \*start && position > 1) {

        printf("Incorrect position...\n");

        free(NN);

        return;

    }

    NN->next = temp->next;

    temp->next = NN;

}

void insert\_end(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    if(\*start == NULL) {

        \*start = NN;

        NN->next = \*start;

    } else {

        Node\* temp = \*start;

        while(temp->next != \*start) {

            temp = temp->next;

        }

        temp->next = NN;

        NN->next = \*start;

    }

}

void delete\_beginning(Node\*\* start) {

    if(\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    Node\* temp = \*start;

    if(temp->next == \*start) {

        free(temp);

        \*start = NULL;

    } else {

        Node\* last = \*start;

        while(last->next != \*start) {

            last = last->next;

        }

        \*start = temp->next;

        last->next = \*start;

        free(temp);

    }

}

void delete\_middle(Node\*\* start, int position) {

    if(\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    if(position == 0) {

        delete\_beginning(start);

        return;

    }

    Node\* temp = \*start;

    for(int i = 0; i < position - 1 && temp->next != \*start; i++) {

        temp = temp->next;

    }

    if(temp->next == \*start) {

        printf("Incorrect position.\n");

        return;

    }

    Node\* NN = temp->next;

    temp->next = NN->next;

    free(NN);

}

void delete\_end(Node\*\* start) {

    if(\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    Node\* temp = \*start;

    if(temp->next == \*start) {

        free(temp);

        \*start = NULL;

        return;

    }

    Node\* prev = NULL;

    while(temp->next != \*start) {

        prev = temp;

        temp = temp->next;

    }

    free(temp);

    prev->next = \*start;

}

void display(Node\* node) {

    if(node == NULL) {

        printf("List is empty.\n");

        return;

    }

    Node\* temp = node;

    do {

        printf("%d -> ", temp->data);

        temp = temp->next;

    } while(temp != node);

    printf("(back to start)\n");

}

int main() {

    Node\* start = NULL;

    int val, position, optn;

    char contn;

    printf("Linked List:\n Insert:\n  1- Beginning.\n  2- Middle.\n  3- End.\n Delete:\n  4- Beginning.\n  5- Middle.\n  6- End.\n  7- Display.\n\n");

    do {

        printf("Enter option: ");

        scanf("%d", &optn);

        switch(optn) {

            case 1:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_beginning(&start, val);

                break;

            case 2:

                printf("Enter value: ");

                scanf("%d", &val);

                printf("Enter position: ");

                scanf("%d", &position);

                insert\_middle(&start, val, position);

                break;

            case 3:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_end(&start, val);

                break;

            case 4:

                delete\_beginning(&start);

                break;

            case 5:

                printf("Enter position: ");

                scanf("%d", &position);

                delete\_middle(&start, position);

                break;

            case 6:

                delete\_end(&start);

                break;

            case 7:

                printf("Linked List display:\n");

                display(start);

                break;

            default:

                printf("Incorrect option...\n");

                break;

        }

        printf("Do you want to continue(Y/N): ");

        scanf(" %c", &contn);

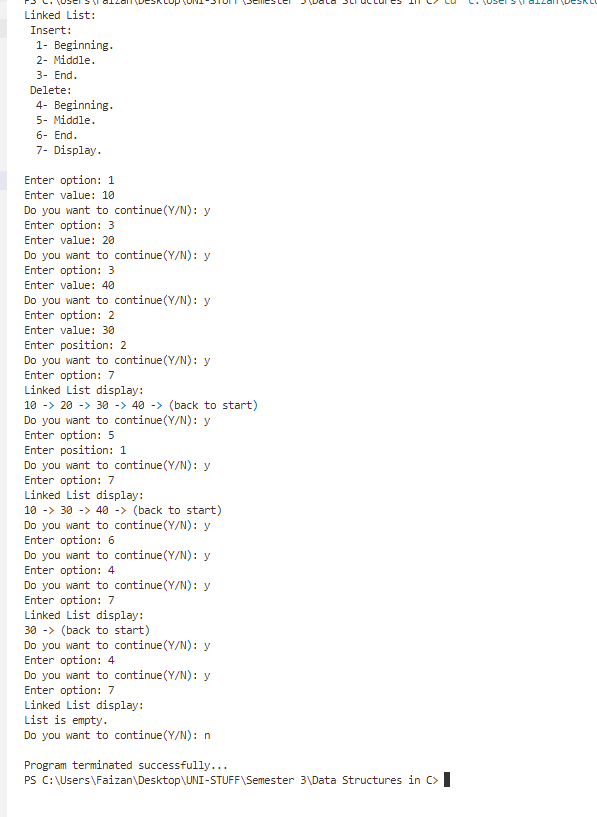
    } while(contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n");

    return 0;

}

**Output:**

****

**Lab 14.2**

**Program:**

Write a program to perform Insertion and Deletion from the beginning, middle, and the end of a circular Doubly Linked List.

**Source code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

    int data;

    struct Node\* next;

    struct Node\* prev;

} Node;

Node\* createNode(int data) {

    Node\* NN = (Node\*)malloc(sizeof(Node));

    if (NN == NULL) {

        printf("List overflow...\n");

        exit(1);

    }

    NN->data = data;

    NN->next = NN;

    NN->prev = NN;

    return NN;

}

void insert\_beginning(Node\*\* start, int data) {

    Node\* NN = createNode(data);

    if (\*start == NULL) {

        \*start = NN;

    } else {

        Node\* last = (\*start)->prev;

        NN->next = \*start;

        NN->prev = last;

        last->next = NN;

        (\*start)->prev = NN;

    }

    \*start = NN;

}

void insert\_middle(Node\*\* start, int data, int position) {

    if (position < 0) {

        printf("Invalid position...\n");

        return;

    }

    if (position == 0) {

        insert\_beginning(start, data);

        return;

    }

    Node\* NN = createNode(data);

    Node\* temp = \*start;

    for (int i = 0; i < position - 1 && temp->next != \*start; i++) {

        temp = temp->next;

    }

    if (temp->next == \*start && position > 1) {

        printf("Incorrect position...\n");

        free(NN);

        return;

    }

    NN->next = temp->next;

    NN->prev = temp;

    temp->next->prev = NN;

    temp->next = NN;

}

void insert\_end(Node\*\* start, int data) {

    if (\*start == NULL) {

        insert\_beginning(start, data);

    } else {

        Node\* NN = createNode(data);

        Node\* last = (\*start)->prev;

        last->next = NN;

        NN->prev = last;

        NN->next = \*start;

        (\*start)->prev = NN;

    }

}

void delete\_beginning(Node\*\* start) {

    if (\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    Node\* temp = \*start;

    if (temp->next == \*start) {

        free(temp);

        \*start = NULL;

    } else {

        Node\* last = temp->prev;

        \*start = temp->next;

        last->next = \*start;

        (\*start)->prev = last;

        free(temp);

    }

}

void delete\_middle(Node\*\* start, int position) {

    if (\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    if (position == 0) {

        delete\_beginning(start);

        return;

    }

    Node\* temp = \*start;

    for (int i = 0; i < position && temp->next != \*start; i++) {

        temp = temp->next;

    }

    if (temp->next == \*start && position > 0) {

        printf("Incorrect position.\n");

        return;

    }

    temp->prev->next = temp->next;

    temp->next->prev = temp->prev;

    free(temp);

}

void delete\_end(Node\*\* start) {

    if (\*start == NULL) {

        printf("List underflow.\n");

        return;

    }

    Node\* last = (\*start)->prev;

    if (last == \*start) {

        free(last);

        \*start = NULL;

    } else {

        Node\* secondLast = last->prev;

        secondLast->next = \*start;

        (\*start)->prev = secondLast;

        free(last);

    }

}

void display(Node\* node) {

    if (node == NULL) {

        printf("List is empty.\n");

        return;

    }

    Node\* temp = node;

    do {

        printf("%d <-> ", temp->data);

        temp = temp->next;

    } while (temp != node);

    printf("(back to start)\n");

}

int main() {

    Node\* start = NULL;

    int val, position, optn;

    char contn;

    printf("Circular Doubly Linked List:\n Insert:\n  1- Beginning.\n  2- Middle.\n  3- End.\n Delete:\n  4- Beginning.\n  5- Middle.\n  6- End.\n  7- Display.\n\n");

    do {

        printf("Enter option: ");

        scanf("%d", &optn);

        switch (optn) {

            case 1:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_beginning(&start, val);

                break;

            case 2:

                printf("Enter value: ");

                scanf("%d", &val);

                printf("Enter position: ");

                scanf("%d", &position);

                insert\_middle(&start, val, position);

                break;

            case 3:

                printf("Enter value: ");

                scanf("%d", &val);

                insert\_end(&start, val);

                break;

            case 4:

                delete\_beginning(&start);

                break;

            case 5:

                printf("Enter position: ");

                scanf("%d", &position);

                delete\_middle(&start, position);

                break;

            case 6:

                delete\_end(&start);

                break;

            case 7:

                printf("Circular Doubly Linked List display:\n");

                display(start);

                break;

            default:

                printf("Incorrect option...\n");

                break;

        }

        printf("Do you want to continue(Y/N): ");

        scanf(" %c", &contn);

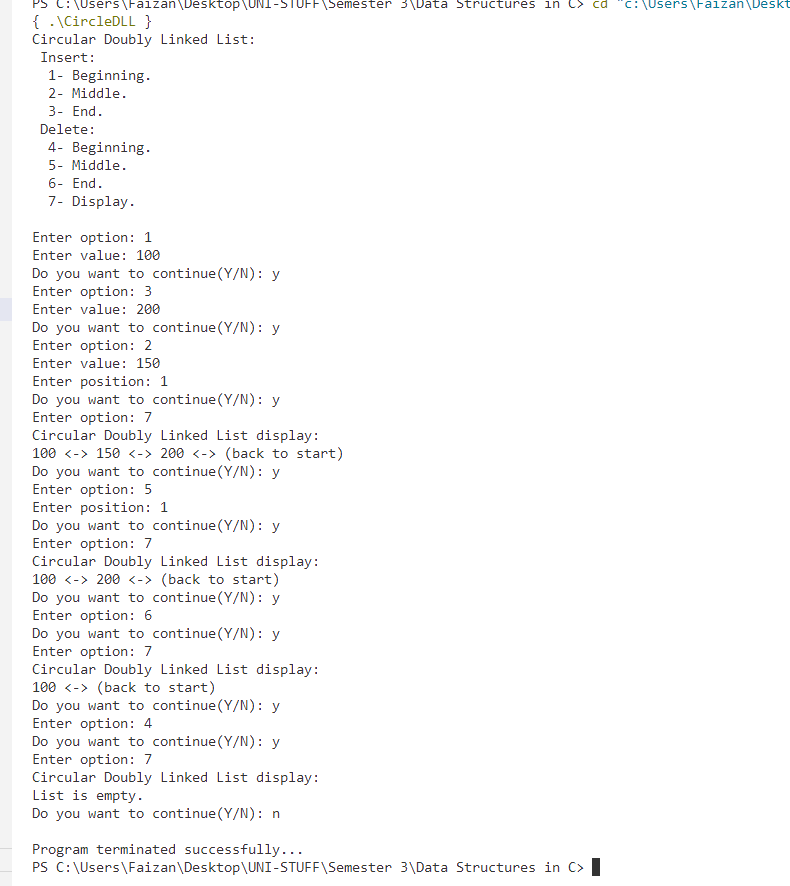
    } while (contn == 'Y' || contn == 'y');

    printf("\nProgram terminated successfully...\n");

    return 0;

}

**Output:**

****