**Delhi Technological University**

Department of Software Engineering



**Data Structures (SE-203)**

**LAB FILE**

**Submitted To :**

Mr. Ankur Narwal

Department of Software Engineering

**Submitted By:**

Aditya Bibhas Sahu

23/SE/009

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Experiment** | **Date** | **Remarks** |
| 1. | Explore the Linux terminal. Check all basic commands for file  handling, compiling c programs, debugging. Install a Linux OS either  on a virtual machine or dual partition.  Write a small program in C to reverse an array, compile using  c and generate a valid output file. | 21/08/24 |  |
| 2. | Create a menu driver program that will take input from the user to:   1. Enter elements in a one-dimensional array. 2. Delete element in a one-dimensional array (have all   conditions, beginning, last, middle index).   1. Find the largest element. 2. Find the smallest element. | 04/09/24 |  |
| 3. | Write a menu driven program to  l. Merge two strings  2 reverse strings  Find a substring and replace it with another string.  All inputs to be taken from the user. | 09/10/24 |  |
| 4. | Write a program to implement character stack using an array Push Popup function using boundry condition Also write paranthesis correctness in a string array. | 16/10/24 |  |
| 5. | Write a program to display, insert and delete element and remove duplicates to a circular queue using menu driven program. Also check for overflow and underflow condition. | 16/10/24 |  |
| 6. | Write a program for displaying , inserting and deleting element to doubly link list. | 23/10/24 |  |
| 7. | Choose a unique expression and store it in a binary tree. Use appropriate tree traversal to generate postfix , prefix and infix. | 23/10/24 |  |

**Experiment-1**

**Aim:** Explore the Linux terminal. Check all basic commands for file handling, compiling c programs, debugging. Install a Linux OS either on a virtual machine or dual partition.

Write a small program in C to reverse an array, compile using c and generate a valid output file.

**Code:**

#include<stdio.h>

// Q1 : Reverse an array

void main(){

int arr[] = {1,23,34,2,112,10,21};

int size=sizeof(arr)/sizeof(arr[0]);

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

int temp=arr[i];

arr[i]=arr[size-i-1];

arr[size-i-1]=temp;

}

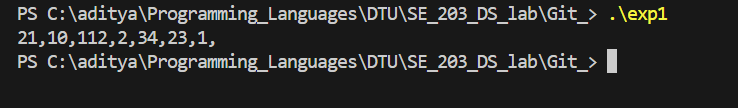
for(int i=0;i<size;i++){

printf("%d,",arr[i]);

}

}

**Output:**



**Experiment-2**

**Aim:** Create a menu driver program that will take input from the user to :

1. Enter elements in a one dimensional array.
2. Delete element in a one dimensional array (have all conditions, beginning,

last, middle index).

1. Find the largest element .
2. Find the smallest element .

**Code:**

#include<stdio.h>

void main(){

printf("Please Enter len of array\n");

int len;

scanf("%d",&len);

int arr[len+1];

int size=0;

int flag=1;

while(flag){

printf("Please Select an input \n");

printf("1. Enter elements in a one dimensional array \n");

printf("2. delete element in a one dimensional array (have all conditions, beginning, last, middle index) \n");

printf("3, Find the largest element \n");

printf("4. Find the smallest element \n");

printf("5. To exit \n");

int a;

scanf("%d",&a);

if(a==1){

printf("Please Enter idx \n");

int idx,num;

scanf("%d",&idx);

printf("Please Enter num \n");

scanf("%d",&num);

for(int i=0;i<size+1;i++){

if(i>=idx){

int temp=arr[i];

arr[i]=num;

num=temp;

}

}

size++;

}

else if(a==2){

printf("Please Enter idx \n");

int idx;

scanf("%d",&idx);

for(int i=0;i<size-1;i++){

if(i>=idx){

arr[i]=arr[i+1];

}

}

size--;

}

else if(a==3){

int max=-1e7;

for(int i=0;i<size;i++){

if(arr[i]>max) max=arr[i];

}

printf("Max = %d\n",max);

}

else if(a==4){

int min=1e7;

for(int i=0;i<size;i++){

if(arr[i]<min) min=arr[i];

}

printf("Min = %d",min);

}

else{

flag=0;

}

for(int i=0;i<size;i++){

printf("%d,",arr[i]);

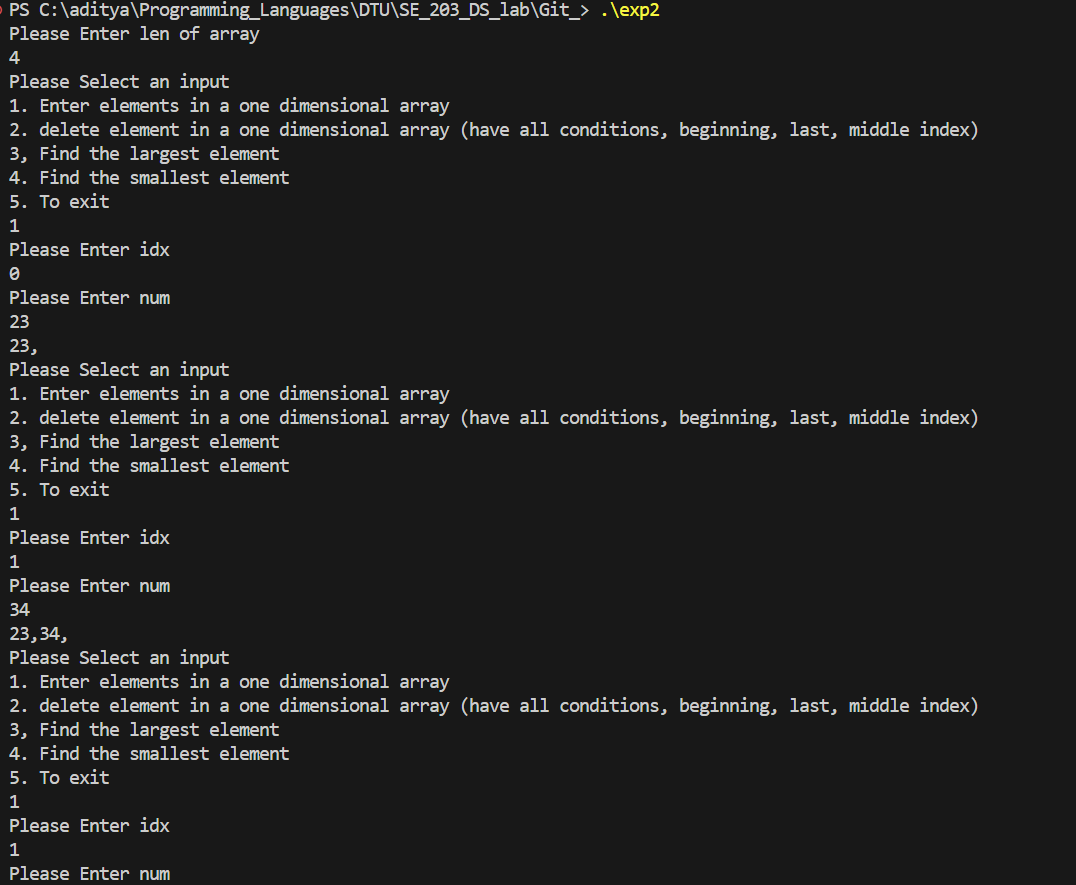
}

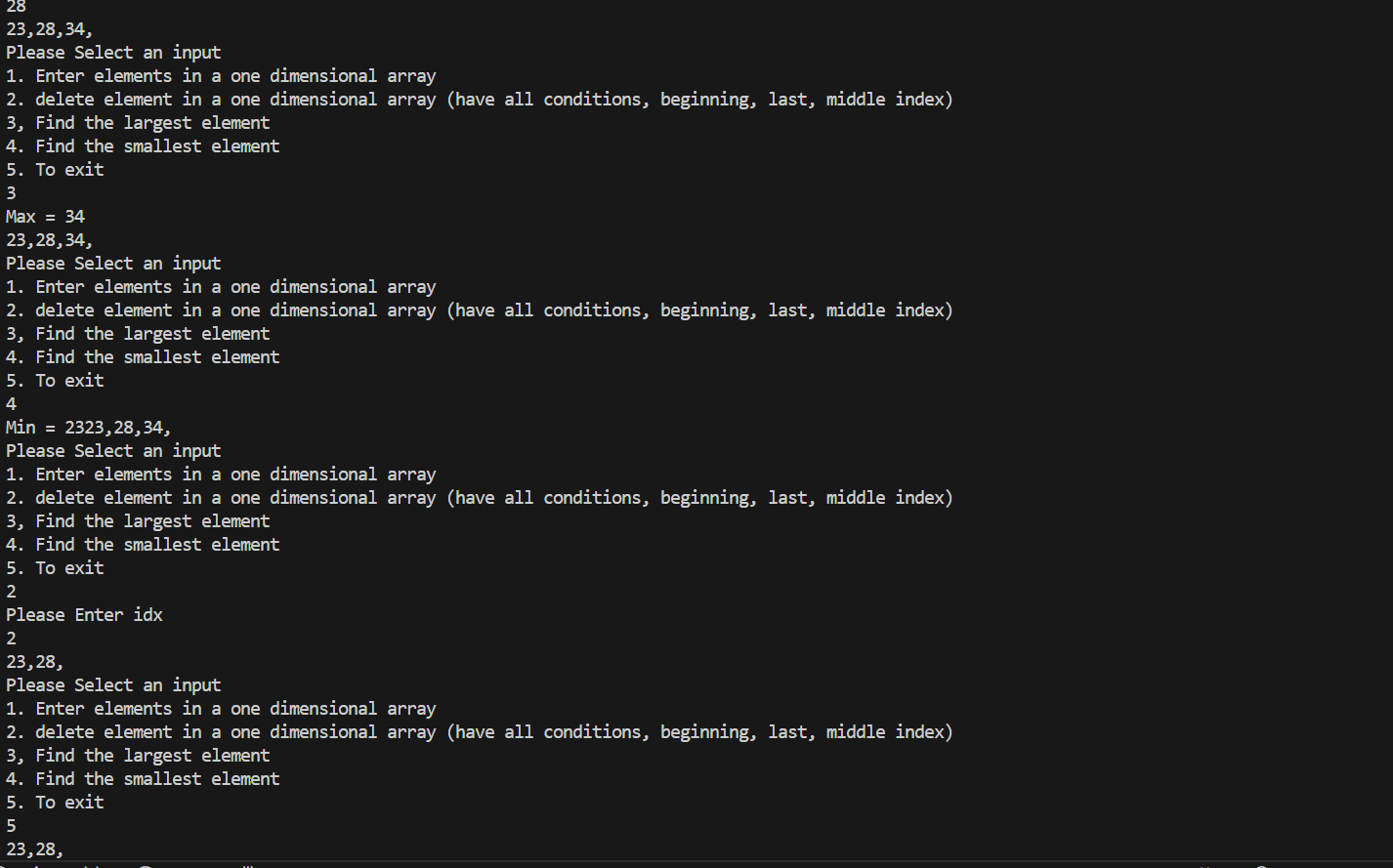
printf("\n");

}

}

**Output:**

****

****

**Experiment-3**

**Aim:** Write a menu driven program to

l. Merge two strings

2 reverse strings

Find a substring and replace it with another string.

All inputs to be taken from the user.

**Code:**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

char\* merge(char\* str1, char\* str2) {

char\* result = (char\*)malloc(strlen(str1) + strlen(str2) + 1);

strcpy(result, str1);

strcat(result, str2);

return result;

}

char\* reverse(char\* str) {

int len = strlen(str);

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

char temp = str[i];

str[i] = str[len - i - 1];

str[len - i - 1] = temp;

}

return str;

}

char\* substring(char\* str, char\* substr, char\* new\_substr) {

static char buffer[1024];

char\* pos;

if (!(pos = strstr(str, substr))) {

return str;

}

strncpy(buffer, str, pos - str);

buffer[pos - str] = '\0';

strcat(buffer, new\_substr);

strcat(buffer, pos + strlen(substr));

strcpy(str, buffer);

return str;

}

int main() {

int flag = 1;

char str1[100], str2[100], str3[100];

while (flag) {

printf("Please Select an option \n");

printf("1. Merge Two strings \n");

printf("2. Reverse strings\n");

printf("3. Find a substring and replace it with another string\n");

printf("4. Exit \n");

int a;

scanf("%d", &a);

getchar();

printf("\n");

if (a == 1) {

printf("Enter String-1: ");

fgets(str1, sizeof(str1), stdin);

str1[strcspn(str1, "\n")] = 0;

printf("Enter String-2: ");

fgets(str2, sizeof(str2), stdin);

str2[strcspn(str2, "\n")] = 0;

char\* result = merge(str1, str2);

printf("The merged string: %s\n", result);

free(result);

} else if (a == 2) {

printf("Enter string to be reversed: ");

fgets(str3, sizeof(str3), stdin);

str3[strcspn(str3, "\n")] = 0;

printf("Reversed string: %s\n", reverse(str3));

} else if (a == 3) {

printf("Enter the main string: ");

fgets(str1, sizeof(str1), stdin);

str1[strcspn(str1, "\n")] = 0;

printf("Enter the substring to find: ");

fgets(str2, sizeof(str2), stdin);

str2[strcspn(str2, "\n")] = 0;

printf("Enter the replacement string: ");

fgets(str3, sizeof(str3), stdin);

str3[strcspn(str3, "\n")] = 0;

printf("The new string: %s\n", substring(str1, str2, str3));

} else {

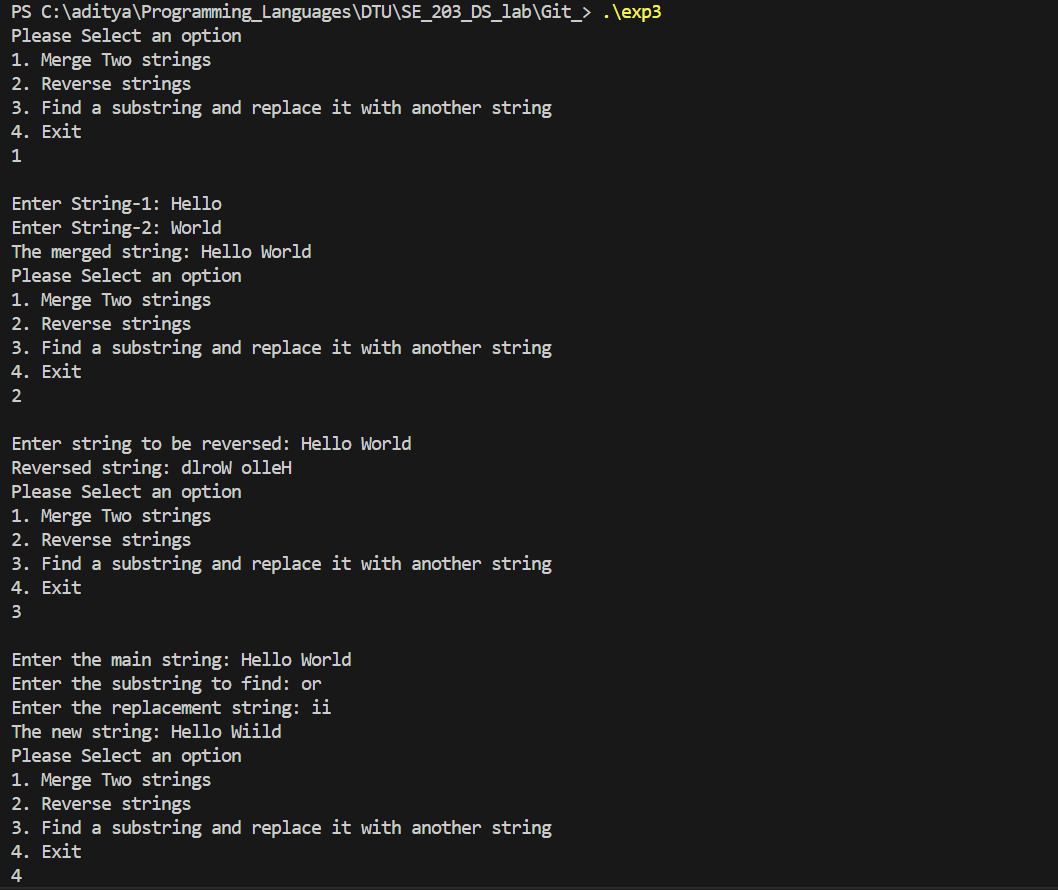
flag = 0;

}

}

}

**Output:**

****

**Experiment-4**

**Aim:** Write a program to implement character stack using an array

Push

Pop functions using boundry condition

Also write paranthesis correctness in a string array.

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 100 // Define maximum size of stack

// Stack structure

struct Stack {

char arr[MAX]; // Stack array to store characters

int top; // Stack top to keep track of the top index

};

// Function to initialize the stack

void initStack(struct Stack\* stack) {

stack->top = -1;

}

// Function to check if the stack is full

bool isFull(struct Stack\* stack) {

return stack->top == MAX - 1;

}

// Function to check if the stack is empty

bool isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

// Function to push a character onto the stack

void push(struct Stack\* stack, char ch) {

if (isFull(stack)) {

printf("Stack overflow! Cannot push %c\n", ch);

return;

}

stack->arr[++stack->top] = ch;

}

// Function to pop a character from the stack

char pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack underflow! Cannot pop\n");

return '\0'; // Return null character if stack is empty

}

return stack->arr[stack->top--];

}

// Function to check for matching parentheses

bool isMatchingPair(char left, char right) {

if (left == '(' && right == ')') return true;

if (left == '{' && right == '}') return true;

if (left == '[' && right == ']') return true;

return false;

}

// Function to check if parentheses in a string are balanced

bool checkParentheses(char str[]) {

struct Stack stack;

initStack(&stack);

for (int i = 0; str[i] != '\0'; i++) {

char ch = str[i];

// If opening bracket, push it to stack

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

push(&stack, ch);

}

// If closing bracket, check for matching opening bracket

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

if (isEmpty(&stack) || !isMatchingPair(pop(&stack), ch)) {

return false; // Unmatched parentheses

}

}

}

// If stack is empty, all parentheses are matched

return isEmpty(&stack);

}

// Main function

int main() {

struct Stack stack;

initStack(&stack);

for(int i=65;i<=69;i++){

push(&stack,(char)i);

}

while(!isEmpty(&stack)){

printf("%c , ",pop(&stack));

}

printf("\n");

char str[MAX];

printf("Enter a string with parentheses: ");

scanf("%s", str);

if (checkParentheses(str)) {

printf("Parentheses are balanced.\n");

} else {

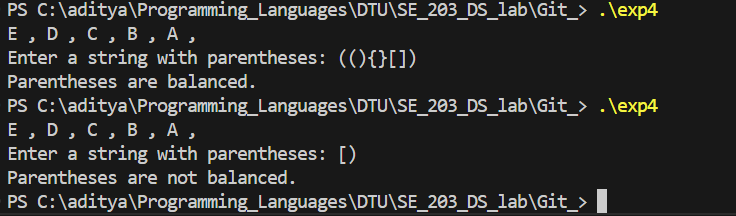
printf("Parentheses are not balanced.\n");

}

return 0;

}

**Output:**

****

**Experiment-5**

**Aim:** Write a program to display, insert and delete element and remove duplicates to a circular queue using menu driven program. Also check for overflow and underflow condition.

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 5  // Define maximum size of the circular queue

// Circular Queue Structure

struct CircularQueue {

    int arr[MAX];

    int front;

    int rear;

};

// Function to initialize the queue

void initQueue(struct CircularQueue\* queue) {

    queue->front = -1;

    queue->rear = -1;

}

// Function to check if the queue is full

bool isFull(struct CircularQueue\* queue) {

    return (queue->front == (queue->rear + 1) % MAX);

}

// Function to check if the queue is empty

bool isEmpty(struct CircularQueue\* queue) {

    return (queue->front == -1);

}

// Function to insert an element into the circular queue

void insert(struct CircularQueue\* queue, int value) {

    if (isFull(queue)) {

        printf("Queue overflow! Cannot insert %d\n", value);

        return;

    }

    if (isEmpty(queue)) {

        queue->front = 0;

    }

    queue->rear = (queue->rear + 1) % MAX;

    queue->arr[queue->rear] = value;

    printf("%d inserted into the queue.\n", value);

}

// Function to delete an element from the circular queue

int delete(struct CircularQueue\* queue) {

    if (isEmpty(queue)) {

        printf("Queue underflow! Cannot delete\n");

        return -1;

    }

    int value = queue->arr[queue->front];

    if (queue->front == queue->rear) {

        // Queue has only one element

        queue->front = queue->rear = -1;

    } else {

        queue->front = (queue->front + 1) % MAX;

    }

    printf("%d deleted from the queue.\n", value);

    return value;

}

// Function to display the queue elements

void display(struct CircularQueue\* queue) {

    if (isEmpty(queue)) {

        printf("Queue is empty.\n");

        return;

    }

    printf("Queue elements: ");

    int i = queue->front;

    while (i != queue->rear) {

        printf("%d ", queue->arr[i]);

        i = (i + 1) % MAX;

    }

    printf("%d\n", queue->arr[i]);  // Display the rear element

}

// Function to remove duplicates from the queue

void removeDuplicates(struct CircularQueue\* queue) {

    if (isEmpty(queue)) {

        printf("Queue is empty. No duplicates to remove.\n");

        return;

    }

    int i = queue->front;

    while (i != queue->rear) {

        int j = (i + 1) % MAX;

        while (j != queue->rear + 1) {

            if (queue->arr[i] == queue->arr[j]) {

                printf("Removing duplicate element %d\n", queue->arr[j]);

                // Shift elements to the left to remove duplicate

                int k = j;

                while (k != queue->rear) {

                    queue->arr[k] = queue->arr[(k + 1) % MAX];

                    k = (k + 1) % MAX;

                }

                queue->rear = (queue->rear - 1 + MAX) % MAX;

            } else {

                j = (j + 1) % MAX;

            }

        }

        i = (i + 1) % MAX;

    }

    printf("Duplicates removed.\n");

}

// Menu-driven program

int main() {

    struct CircularQueue queue;

    initQueue(&queue);

    int choice, value;

    while (1) {

        printf("\n\*\*\* Circular Queue Menu \*\*\*\n");

        printf("1. Insert\n");

        printf("2. Delete\n");

        printf("3. Display\n");

        printf("4. Remove Duplicates\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to insert: ");

                scanf("%d", &value);

                insert(&queue, value);

                break;

            case 2:

                delete(&queue);

                break;

            case 3:

                display(&queue);

                break;

            case 4:

                removeDuplicates(&queue);

                break;

            case 5:

                printf("Exiting program.\n");

                return 0;

            default:

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

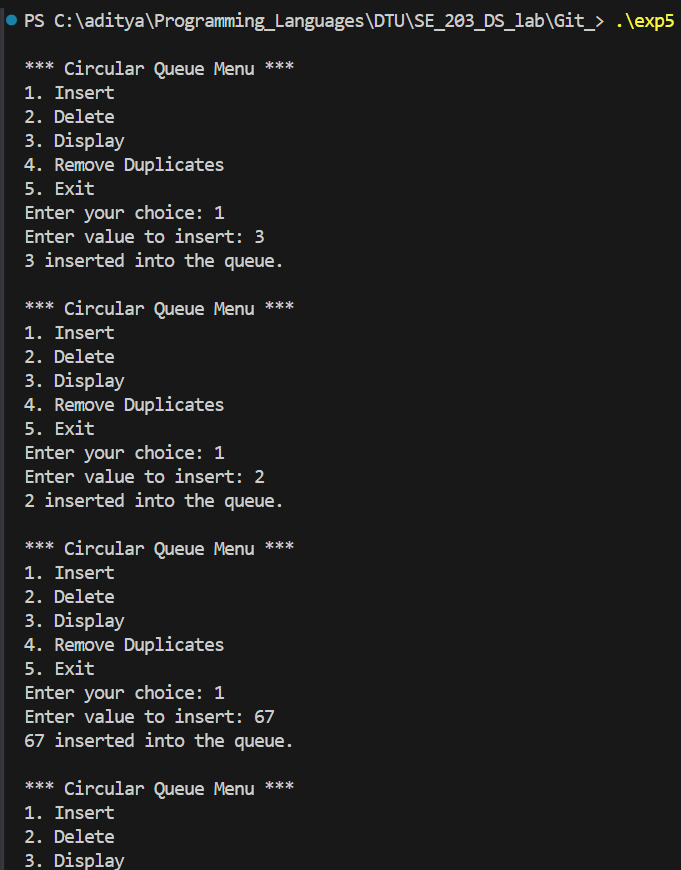
        }

    }

    return 0;

}

**Output:**



**Experiment-6**

**Aim:** write a program for displaying, inserting and deleting element to doubly link list

**Code:**

#include <stdio.h>

#include <stdlib.h>

// Doubly linked list node structure

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

// Function to create a new node

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// Function to display the doubly linked list

void display(struct Node\* head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

printf("Doubly linked list elements: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

// Function to insert a node at the beginning of the list

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

newNode->next = \*head;

(\*head)->prev = newNode;

\*head = newNode;

}

printf("%d inserted at the beginning of the list.\n", data);

}

// Function to insert a node at the end of the list

void insertAtEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

printf("%d inserted at the end of the list.\n", data);

}

// Function to delete a node from the list by value

void deleteNode(struct Node\*\* head, int data) {

if (\*head == NULL) {

printf("The list is empty. Cannot delete %d.\n", data);

return;

}

struct Node\* temp = \*head;

// Search for the node to delete

while (temp != NULL && temp->data != data) {

temp = temp->next;

}

// If the node to delete is not found

if (temp == NULL) {

printf("Element %d not found in the list.\n", data);

return;

}

// If the node to delete is the head node

if (temp == \*head) {

\*head = temp->next;

if (\*head != NULL) {

(\*head)->prev = NULL;

}

} else {

// If the node to delete is in the middle or end

if (temp->prev != NULL) {

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

}

if (temp->next != NULL) {

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

}

}

free(temp);

printf("Element %d deleted from the list.\n", data);

}

// Menu-driven program for doubly linked list operations

int main() {

struct Node\* head = NULL;

int choice, value;

while (1) {

printf("\n\*\*\* Doubly Linked List Menu \*\*\*\n");

printf("1. Insert at Beginning\n");

printf("2. Insert at End\n");

printf("3. Delete by Value\n");

printf("4. Display List\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at the beginning: ");

scanf("%d", &value);

insertAtBeginning(&head, value);

break;

case 2:

printf("Enter value to insert at the end: ");

scanf("%d", &value);

insertAtEnd(&head, value);

break;

case 3:

printf("Enter value to delete: ");

scanf("%d", &value);

deleteNode(&head, value);

break;

case 4:

display(head);

break;

case 5:

printf("Exiting program.\n");

return 0;

default:

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

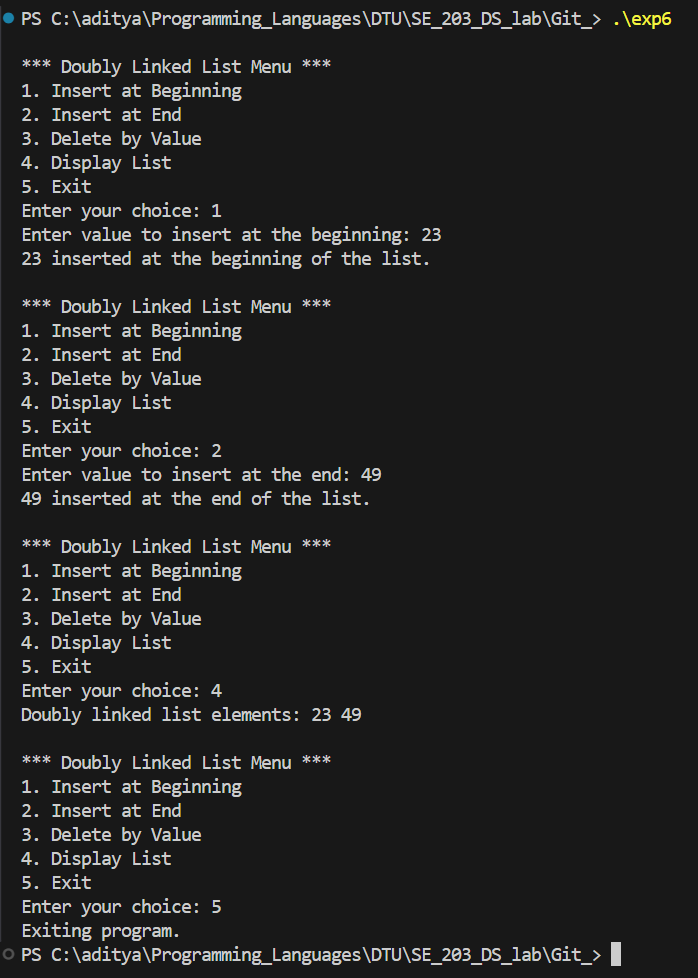
}

}

return 0;

}

**Output:**



**Experiment-7**

**Aim:** Choose a unique expression and store it in a binary tree. Use appropriate tree traversal to generate postfix , prefix and infix

**Code:**

#include <stdio.h>

#include <stdlib.h>

// Structure for a tree node

struct Node {

char data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new tree node

struct Node\* createNode(char data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Inorder traversal (Infix expression)

void inorder(struct Node\* node) {

if (node == NULL)

return;

// Traverse the left subtree

inorder(node->left);

// Visit the root (current node)

printf("%c ", node->data);

// Traverse the right subtree

inorder(node->right);

}

// Preorder traversal (Prefix expression)

void preorder(struct Node\* node) {

if (node == NULL)

return;

// Visit the root (current node)

printf("%c ", node->data);

// Traverse the left subtree

preorder(node->left);

// Traverse the right subtree

preorder(node->right);

}

// Postorder traversal (Postfix expression)

void postorder(struct Node\* node) {

if (node == NULL)

return;

// Traverse the left subtree

postorder(node->left);

// Traverse the right subtree

postorder(node->right);

// Visit the root (current node)

printf("%c ", node->data);

}

// Main function to demonstrate the traversals

int main() {

// Manually constructing the binary tree for expression: ((3 + 2) \* (5 - 4))

// Creating nodes for the expression

struct Node\* root = createNode('\*');

root->left = createNode('+');

root->right = createNode('-');

root->left->left = createNode('3');

root->left->right = createNode('2');

root->right->left = createNode('5');

root->right->right = createNode('4');

printf("Infix Expression (Inorder Traversal): ");

inorder(root);

printf("\n");

printf("Prefix Expression (Preorder Traversal): ");

preorder(root);

printf("\n");

printf("Postfix Expression (Postorder Traversal): ");

postorder(root);

printf("\n");

return 0;

}

**Output:**

