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>

// 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;
   }
```

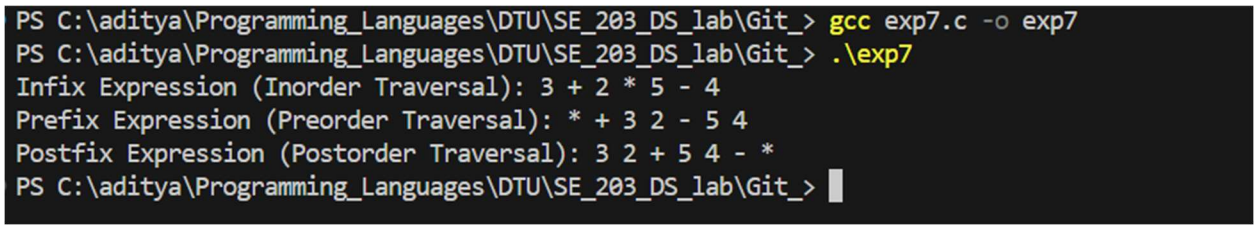
```
    printf("\n");
}

// 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:


PS C:\aditya\Programming_Languages\DTU\SE_203_DS_lab\Git_> gcc exp7.c -o exp7
PS C:\aditya\Programming_Languages\DTU\SE_203_DS_lab\Git_> .\exp7
Infix Expression (Inorder Traversal): 3 + 2 * 5 - 4
Prefix Expression (Preorder Traversal): * + 3 2 - 5 4
Postfix Expression (Postorder Traversal): 3 2 + 5 4 - *
PS C:\aditya\Programming_Languages\DTU\SE_203_DS_lab\Git_>