**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Shahabad Daulatpur, Bawana Road, Delhi-110042



**DATA STRUCTURES LABORATORY**

**(SE-203n)**

**SUBMITTED TO:**

Mr. Ankur Narwal

Department of Software Engineering

**SUBMITTED BY:**

Aditya Kumar

23/SE/011

BTech SE 2nd year

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Objective** | **Date** | **Sign.** |
| 1. | To take input from the user an array of integers and reverse the array. | 21/08/24 |  |
| 2. | To perform various operations on a 1-Dimensional array using a menu driven program:   1. Insert an element into the array. 2. Delete an element into the array. 3. Finding the largest element in the array. 4. Finding the smallest element in the array. | 04/09/24 |  |
| 3. | To perform various string operations using a menu driven program:   1. Merging two strings. 2. Reversing a string. 3. Finding a substring in a given string and replacing it with another string. | 09/10/24 |  |
| 4. | Write a program to implement a character stack using an array and implement:   1. Push() operation. 2. Pop() operation. 3. Parenthesis correctness of a string array. | 16/10/24 |  |
| 5. | To implement a circular queue using array and perform various operations using a menu driven program:   1. Insert elements in the queue at the tail. 2. Delete elements from the queue. 3. Remove duplicates from the queue.   Check for overflow and underflow conditions. | 16/10/24 |  |
| 6. | Write a menu driven program to create a Doubly Linked List and perform:   1. Insert elements into the Doubly Linked List. 2. Delete elements from the Doubly Linked List. 3. Display the Doubly Linked List. | 23/10/24 |  |
| 7. | Choose an expression and store it in a binary tree. Use appropriate tree traversal to:   1. generate postfix expression. 2. generate prefix expression. 3. generate infix expression.   For the same. | 23/10/24 |  |

**EXPERIMENT – 1**

**AIM:** To take input from the user an array of integers and reverse the array.

**CODE:**

#include <stdio.h>

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

    for(int i = 0; i < n/2; i++) {

        int temp = arr[i];

        arr[i] = arr[n-1-i];

        arr[n-1-i] = temp;

    }

}

int main(void) {

    int n;

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

    scanf("%d", &n);

    int arr[n];

    printf("Enter elements space seperated: ");

    for(int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    reverse(arr, n);

    printf("Elements in array after reversal: \n");

    for(int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

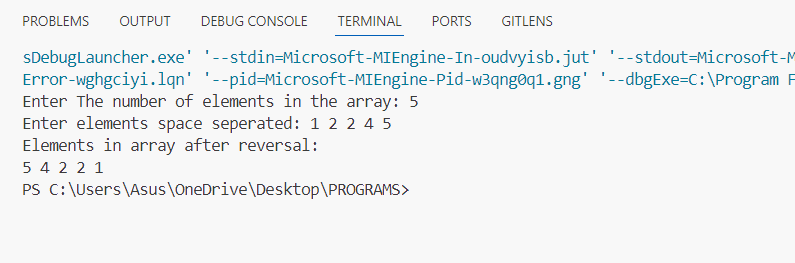
    }

    printf("\n");

    return 0;

}

**OUTPUT:**

****

**EXPERIMENT – 2**

**AIM:** To perform various operations on a 1-Dimensional array using a menu driven program:

1. Insert an element into the array.
2. Delete an element into the array.
3. Finding the largest element in the array.
4. Finding the smallest element in the array.

**CODE:**

#include <stdio.h>

#include <limits.h>

int largest(int\* arr, int n) {

    int maxi = INT\_MIN;

    for(int i = 0; i < n; i++) {

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

    }

    return maxi;

}

int smallest(int\* arr, int n) {

    int mini = INT\_MAX;

    for(int i = 0; i < n; i++) {

        if (mini > arr[i] && arr[i] != -1) mini = arr[i];

    }

    return mini;

}

void insert(int \*arr, int n, int pos, int x) {

    int i = 0;

    while(i < pos) i++;

    int next = arr[i];

    arr[i] = x;

    int temp;

    while(i + 1 < n) {

        temp = arr[i+1];

        arr[i+1] = next;

        next = temp;

        i++;

    }

}

void delete\_from(int \*arr, int n, int pos) {

    int i = 0;

    while(i < pos) i++;

    int temp;

    while(i + 1 < n) {

        arr[i] = arr[i+1];

        i++;

    }

}

void display(int \*arr, int n) {

    printf("[ ");

    for(int i = 0; i < n; i++ ) {

        printf("%d ", arr[i]);

    }

    printf("]  "); printf("size = %d\n", n);

}

int main(void) {

    int y = 1;

    int choice;

    int n = 0;

    int MAX\_SIZE = 10;

    int arr[MAX\_SIZE];

    for(int i =0; i < MAX\_SIZE; i++) {

        arr[i] = -1;

    }

    printf("MENU DRIVER PROGRAM TO PERFORM OPERATIONS ON A ONE DIMENSIONAL ARRAY: \n");

    while(y) {

        printf("Choices: \n1. Initialise array. \n2. Insert elements into the array. \n3. Delete element from array.\n4. Find the smallest element in the array. \n5. Find the largest element in the array.\n");

        printf("choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1:

                printf("Enter the number of elements :");

                scanf("%d", &n);

                printf("\nEnter elements space seperated :");

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

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

                }

                printf("An array of size %d has been initialised. \n", n);

                display(arr, n);

                break;

            case 2:

                int position\_insert;

                printf("Enter the position at which you want to insert element in the array :");

                scanf("%d", &position\_insert);

                if (position\_insert > MAX\_SIZE - 1 || position\_insert < 0) {printf("Invalid Position Passed. "); break;}

                printf("Enter the element that is to be inserted in the array :");

                int element;

                scanf("%d", &element);

                insert(arr, MAX\_SIZE, position\_insert, element);

                n = n+1;

                display(arr, n);

                printf("\n");

                break;

            case 3:

                int position\_delete;

                printf("Enter the position at which you want to delete element from the array :");

                scanf("%d", &position\_delete);

                if (position\_delete > MAX\_SIZE - 1 || position\_delete < 0) {printf("Invalid Position Passed. "); break;}

                delete\_from(arr, MAX\_SIZE, position\_delete);

                n = n-1;

                display(arr, n);

                printf("\n");

                break;

            case 5:

                printf("Maximum element in the array is: ");

                printf("%d\n", largest(arr, n));

                break;

            case 4:

                printf("Minimum element in the array is: ");

                printf("%d\n", smallest(arr, n));

                break;

            default:

                printf("Invalid choice inputted.");

                break;

        }

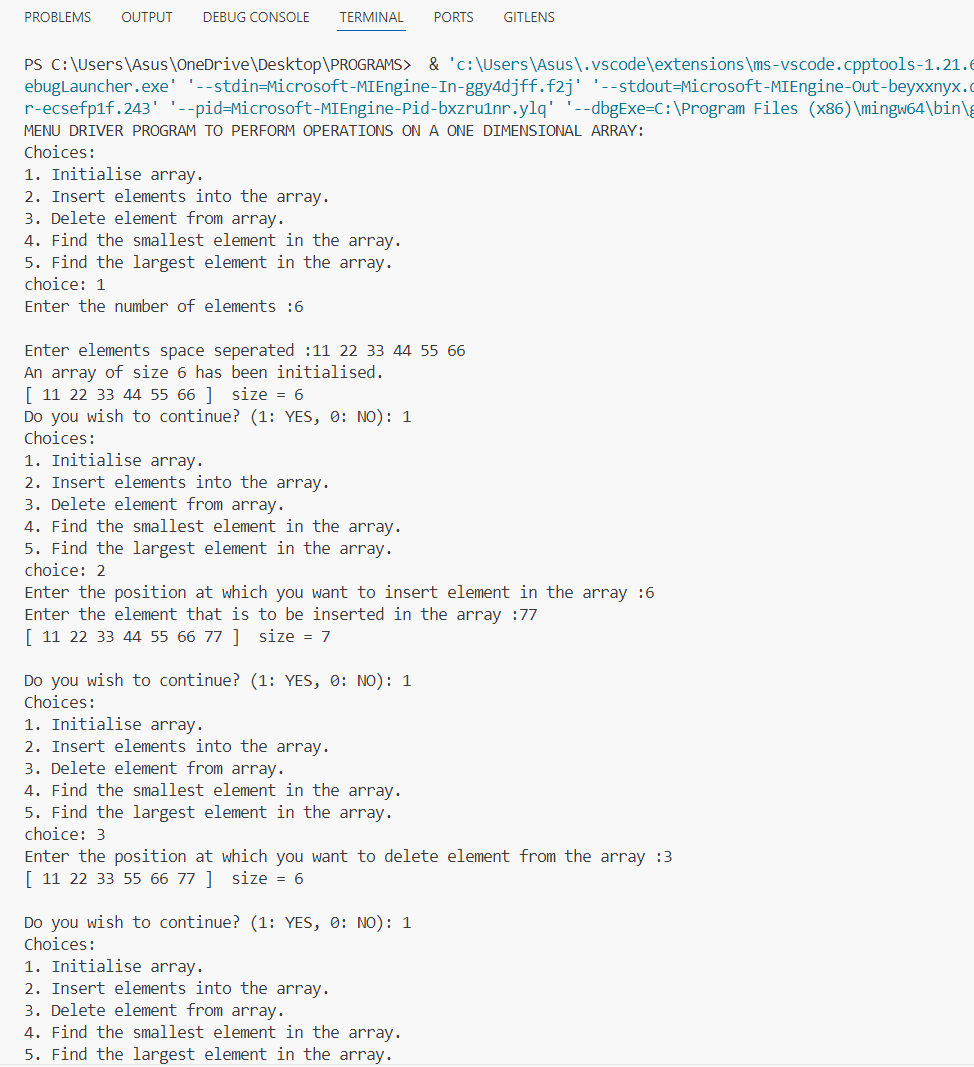
        printf("Do you wish to continue? (1: YES, 0: NO): ");

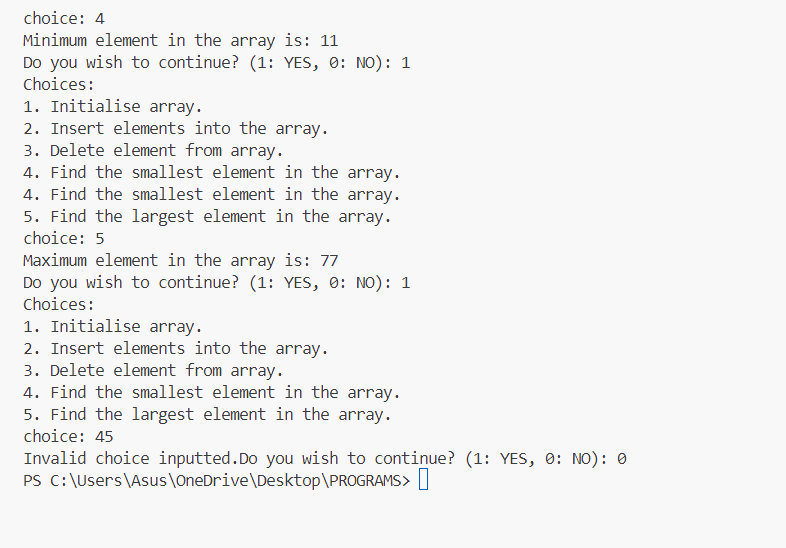
        scanf("%d", &y);

    }

}

**OUTPUT:**

****

****

**EXPERIMENT – 3**

**AIM:** To perform various string operations using a menu driven program:

1. Merging two strings.
2. Reversing a string.
3. Finding a substring in a given string and replacing it with another string.

**CODE:**

#include <stdio.h>

#include <string.h>

void merge\_strings() {

    char s1[100];

    char s2[100];

    printf("Enter the string-1: ");

    scanf("%s", s1);

    printf("Enter the string-2: ");

    scanf("%s", s2);

    printf("Merging string-2 into string-1..\n");

    int i = strlen(s1);

    int j;

    for(j = i; j<i+strlen(s2); j++) {

        s1[j] = s2[j-i];

    }

    s1[j] = '\0';

    printf("Resultant string is: %s", s1);

}

void reverse\_string() {

    char s1[100];

    printf("Enter the string that needs to be reversed: ");

    scanf("%s", s1);

    for(int i = 0; i < strlen(s1)/2; i++) {

        char temp = s1[i];

        s1[i] = s1[strlen(s1)-i-1];

        s1[strlen(s1)-i-1] = temp;

    }

    printf("Resultant string after reversal is: %s", s1);

}

void substringfindreplace(void) {

    char m[100]; // mainstring;

    printf("Enter the main string: ");

    scanf("%s", m);

    char find[100];

    printf("Enter the substring that needs to be replaced: ");

    scanf("%s", find);

    int  i= 0, j = 0;

    int start\_pos = -1, end\_pos = -1;

    for(int i = 0; i < strlen(m); i++) {

        if (m[i] == find[j]) {

            start\_pos = i;

            int dummy\_i = i;

            while(dummy\_i < strlen(m) && m[dummy\_i] == find[j]) {

                dummy\_i++; j++;

            }

            if (j == strlen(find))  {

                end\_pos = dummy\_i;

                break;

            }

            else j = 0;  start\_pos = -1;

        }

    }

    if (start\_pos == -1) {printf("Pattern/ substring not found in the given string."); return;}

    printf("Pattern found in main string \"%s\"  at position: %d", m, start\_pos);

    char replace[100];

    printf("\nEnter the string that should replace the pattern: ");

    scanf("%s", replace);

    char newstring[100];

    int k = 0;

    i = 0;

    while(i != start\_pos) {

        newstring[k] = m[i];

        k++; i++;

    }

    j = 0;

    while(j < strlen(replace)) {

        newstring[k] = replace[j];

        j++; k++;

    }

    i = end\_pos;

    while(m[i] != '\0') {

        newstring[k] = m[i];

        k++;i++;

    }

    newstring[k] = '\0';

    printf("New string after replacing the pattern: %s", newstring);

}

int main(void) {

    printf("PROGRAM-3: MENU DRIVEN PROGRAM FOR STRINGS.\n\n");

    char y = 'Y';

    do {

        printf("1. Merge two strings\n2. Reverse strings.\n3. Find a substring and replace it with another string.\n");

        int choice;

        printf("Enter choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                merge\_strings();

                break;

            case 2:

                reverse\_string();

                break;

            case 3:

                substringfindreplace();

                break;

            default:

                printf("Invalid choice entered.\n");

                break;

        }

        // Fix: Add a space before %c to consume the newline left by previous input

        printf("\n\nContinue? (Y/N): ");

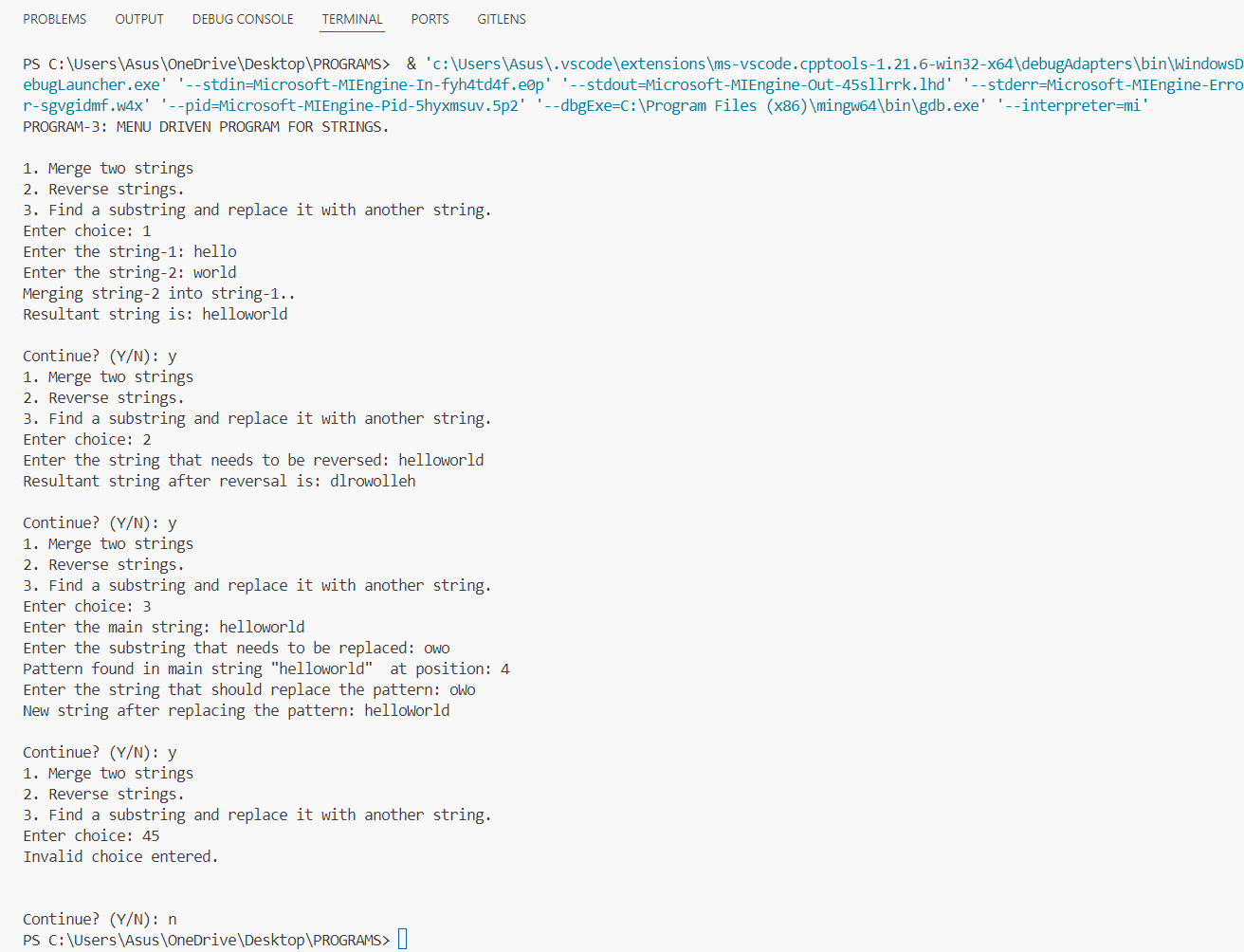
        scanf(" %c", &y);  // Notice the space before %c

    } while (y == 'Y' || y == 'y');  // Allow 'y' as well for case-insensitivity

    return 0;

}

**OUTPUT:**

****

**EXPERIMENT – 4**

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

1. Push() operation.
2. Pop() operation.
3. Parenthesis correctness of a string array.

**CODE:**

#include <stdio.h>

#define STACK\_SIZE 100

char stackArray[STACK\_SIZE];

int stackIndex = -1;

void addToStack(char ch) {

    if (stackIndex < STACK\_SIZE - 1) {

        stackArray[++stackIndex] = ch;

    }

}

char removeFromStack() {

    if (stackIndex >= 0) {

        return stackArray[stackIndex--];

    }

    return '\0';

}

int isPairMatched(char opening, char closing) {

    if (opening == '(' && closing == ')') return 1;

    if (opening == '{' && closing == '}') return 1;

    if (opening == '[' && closing == ']') return 1;

    return 0;

}

int validateParentheses(char expression[]) {

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

        if (expression[i] == '(' || expression[i] == '{' || expression[i] == '[') {

            addToStack(expression[i]);

        } else if (expression[i] == ')' || expression[i] == '}' || expression[i] == ']') {

            char topChar = removeFromStack();

            if (!isPairMatched(topChar, expression[i])) {

                return 0;

            }

        }

    }

    return (stackIndex == -1);

}

int main() {

    char expression[STACK\_SIZE];

    printf("Input your expression: ");

    scanf("%s", expression);

    if (validateParentheses(expression)) {

        printf("Parentheses are balanced\n");

    } else {

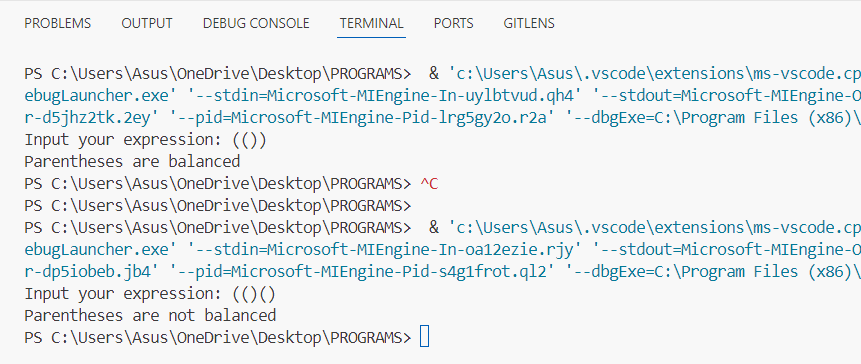
        printf("Parentheses are not balanced\n");

    }

    return 0;

}

**OUTPUT:**

****

**EXPERIMENT – 5**

**AIM:** To implement a circular queue using array and perform various operations using a menu driven program:

1. Insert elements in the queue at the tail.
2. Delete elements from the queue.
3. Remove duplicates from the queue.

Check for overflow and underflow conditions.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5  // Define maximum capacity for the circular queue

// Structure to represent the circular queue

struct CQueue {

    int elements[MAX\_SIZE];

    int head;

    int tail;

};

// Function to initialize the queue

void initializeQueue(struct CQueue \*queue) {

    queue->head = -1;

    queue->tail = -1;

}

// Function to check if the queue is full

int isQueueFull(struct CQueue \*queue) {

    return ((queue->head == 0 && queue->tail == MAX\_SIZE - 1) || (queue->head == queue->tail + 1));

}

// Function to check if the queue is empty

int isQueueEmpty(struct CQueue \*queue) {

    return (queue->head == -1);

}

// Function to add an element at the tail of the queue

void addToTail(struct CQueue \*queue, int data) {

    if (isQueueFull(queue)) {

        printf("Queue Overflow!\n");

        return;

    }

    if (queue->head == -1)

        queue->head = 0;

    queue->tail = (queue->tail + 1) % MAX\_SIZE;

    queue->elements[queue->tail] = data;

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

}

// Function to remove an element from the head of the queue

int removeFromHead(struct CQueue \*queue) {

    int removedElement;

    if (isQueueEmpty(queue)) {

        printf("Queue Underflow!\n");

        return -1;

    }

    removedElement = queue->elements[queue->head];

    if (queue->head == queue->tail) {

        queue->head = -1;

        queue->tail = -1;

    } else {

        queue->head = (queue->head + 1) % MAX\_SIZE;

    }

    printf("Removed element: %d\n", removedElement);

    return removedElement;

}

// Function to show the elements in the queue

void showQueue(struct CQueue \*queue) {

    if (isQueueEmpty(queue)) {

        printf("Queue is empty.\n");

        return;

    }

    printf("Queue elements: ");

    for (int i = queue->head; i != queue->tail; i = (i + 1) % MAX\_SIZE) {

        printf("%d ", queue->elements[i]);

    }

    printf("%d\n", queue->elements[queue->tail]);

}

// Function to display the front element of the queue

void showFrontElement(struct CQueue \*queue) {

    if (isQueueEmpty(queue)) {

        printf("Queue is empty!\n");

    } else {

        printf("Front element: %d\n", queue->elements[queue->head]);

    }

}

// Function to remove duplicate elements from the queue

void deleteDuplicates(struct CQueue \*queue) {

    if (isQueueEmpty(queue)) {

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

        return;

    }

    int i, j, k;

    for (i = queue->head; i != queue->tail; i = (i + 1) % MAX\_SIZE) {

        j = (i + 1) % MAX\_SIZE;

        while (j != (queue->tail + 1) % MAX\_SIZE) {

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

                for (k = j; k != queue->tail; k = (k + 1) % MAX\_SIZE) {

                    queue->elements[k] = queue->elements[(k + 1) % MAX\_SIZE];

                }

                queue->tail = (queue->tail - 1 + MAX\_SIZE) % MAX\_SIZE;

            } else {

                j = (j + 1) % MAX\_SIZE;

            }

        }

    }

    printf("Duplicates removed.\n");

}

// Main program for the menu-driven program

int main() {

    struct CQueue queue;

    initializeQueue(&queue);

    int action, element;

    do {

        printf("\nMenu:\n");

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

        printf("2. Remove from Head\n");

        printf("3. Display Queue\n");

        printf("4. Show Front Element\n");

        printf("5. Remove Duplicates\n");

        printf("6. Quit\n");

        printf("Select an option: ");

        scanf("%d", &action);

        switch (action) {

            case 1:

                printf("Enter a value to insert: ");

                scanf("%d", &element);

                addToTail(&queue, element);

                break;

            case 2:

                removeFromHead(&queue);

                break;

            case 3:

                showQueue(&queue);

                break;

            case 4:

                showFrontElement(&queue);

                break;

            case 5:

                deleteDuplicates(&queue);

                break;

            case 6:

                printf("Exiting program.\n");

                break;

            default:

                printf("Invalid selection.\n");

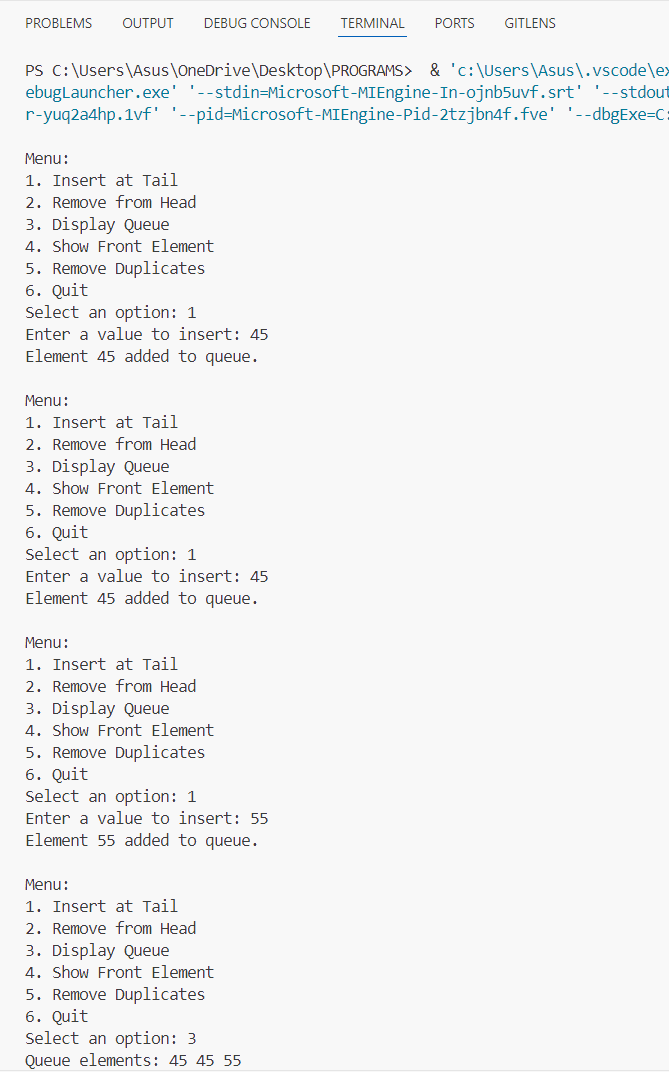
        }

    } while (action != 6);

    return 0;

}

**OUTPUT:**

****

****

**EXPERIMENT – 6**

**AIM:** Write a menu driven program to create a Doubly Linked List and perform:

1. Insert elements into the Doubly Linked List.
2. Delete elements from the Doubly Linked List.
3. Display the Doubly Linked List.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

// Defining a node for the doubly linked list

struct Node {

    int value;

    struct Node\* next;

    struct Node\* prev;

};

// Function to create a new node

struct Node\* createNewNode(int val) {

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

    newNode->value = val;

    newNode->next = NULL;

    newNode->prev = NULL;

    return newNode;

}

// Insert node at the end of the list

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

    struct Node\* newNode = createNewNode(val);

    if (\*head == NULL) {

        \*head = newNode;

    } else {

        struct Node\* temp = \*head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

}

// Insert node at the beginning of the list

void insertAtBegin(struct Node\*\* head, int val) {

    struct Node\* newNode = createNewNode(val);

    if (\*head != NULL) {

        newNode->next = \*head;

        (\*head)->prev = newNode;

    }

    \*head = newNode;

}

// Insert node in the middle at a specific position

void insertNode(struct Node\*\* head, int val, int pos) {

    if (pos == 1) {

        insertAtBegin(head, val);

        return;

    }

    struct Node\* newNode = createNewNode(val);

    struct Node\* temp = \*head;

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

        temp = temp->next;

    }

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

        insertAtEnd(head, val);

    } else {

        newNode->next = temp->next;

        newNode->prev = temp;

        if (temp->next != NULL) {

            temp->next->prev = newNode;

        }

        temp->next = newNode;

    }

}

// Delete node at the end

void deleteAtEnd(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = \*head;

    if (temp->next == NULL) {

        free(temp);

        \*head = NULL;

    } else {

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->prev->next = NULL;

        free(temp);

    }

}

// Delete node at the beginning

void deleteAtBegin(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = \*head;

    \*head = (\*head)->next;

    if (\*head != NULL) {

        (\*head)->prev = NULL;

    }

    free(temp);

}

// Delete node in the middle at a specific position

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

    if (\*head == NULL) {

        printf("List is empty.\n");

        return;

    }

    if (pos == 1) {

        deleteAtBegin(head);

        return;

    }

    struct Node\* temp = \*head;

    for (int i = 1; i < pos && temp != NULL; i++) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Position out of range.\n");

        return;

    }

    if (temp->next != NULL) {

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

    }

    if (temp->prev != NULL) {

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

    }

    free(temp);

}

// Display the elements of the list

void display(struct Node\* head) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = head;

    while (temp != NULL) {

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

        temp = temp->next;

    }

    printf("X\n");

}

// Main function to drive the menu

int main() {

    struct Node\* head = NULL;

    int choice, val, pos;

    do {

        printf("\nMenu:\n");

        printf("1. Insert Into the List.\n");

        printf("2. Delete from List.\n");

        printf("3. Display List.\n");

        printf("4. Exit.\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to insert: ");

                scanf("%d", &val);

                printf("Enter position to insert at: ");

                scanf("%d", &pos);

                insertNode(&head, val, pos);

                break;

            case 2:

                printf("Enter position to delete from: ");

                scanf("%d", &pos);

                deleteNode(&head, pos);

                break;

            case 3:

                display(head);

                break;

            case 4:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice. Try again.\n");

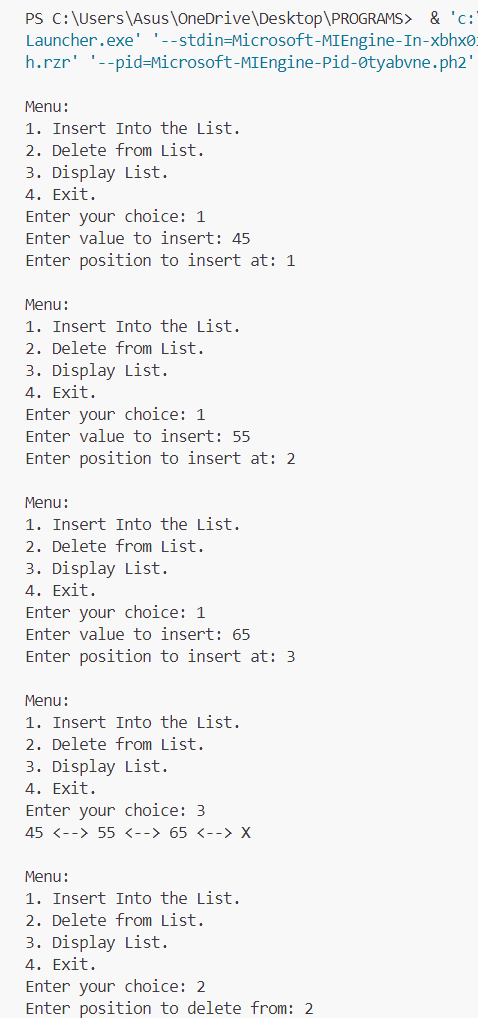
        }

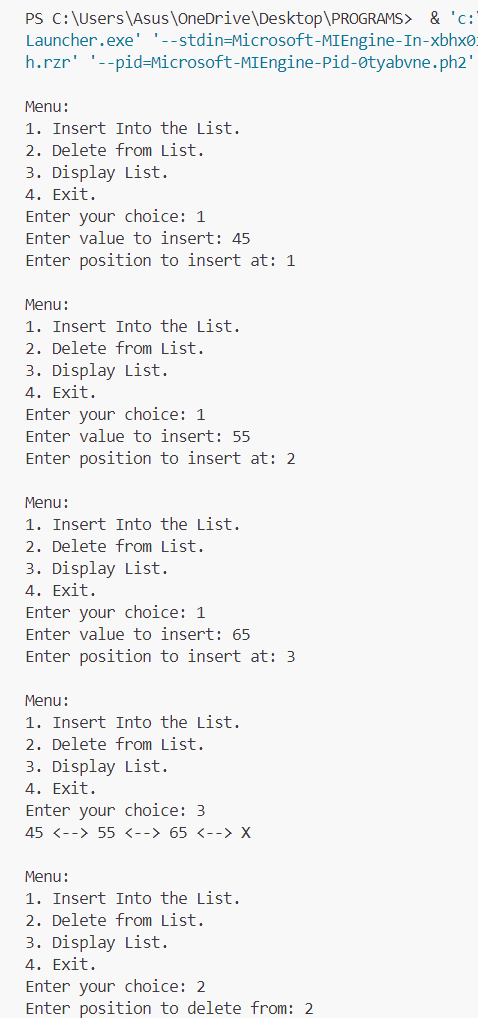
    } while (choice != 4);

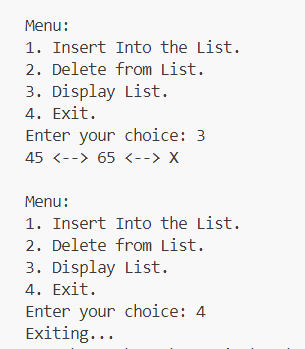
    return 0;

}

**OUTPUT:**

****

****

****

**EXPERIMENT – 7**

**AIM:** Choose an expression and store it in a binary tree. Use appropriate tree traversal to:

1. generate postfix expression.
2. generate prefix expression.
3. generate infix expression.

For the same.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#define exp "A\*B-C/D"

typedef struct TreeNode {

    char val;

    struct TreeNode\* left;

    struct TreeNode\* right;

} TreeNode;

TreeNode\* constructNode(char x) {

    TreeNode\* node = (TreeNode\*) malloc(sizeof(TreeNode));

    node->left = NULL; node->right = NULL;

    node->val = x;

    return node;

}

void preorder(TreeNode\* root) {

    if (root == NULL) return;

    printf("%c", root->val);

    preorder(root->left);

    preorder(root->right);

}

void inorder(TreeNode\* root) {

    if (root == NULL) return;

    inorder(root->left);

    printf("%c", root->val);

    inorder(root->right);

}

void postorder(TreeNode\* root) {

    if (root == NULL) return;

    postorder(root->left);

    postorder(root->right);

    printf("%c", root->val);

}

TreeNode\* constructExp(void) {

    TreeNode\* root = constructNode('-');

    root->left = constructNode('\*');

    root->right = constructNode('/');

    root->left->left = constructNode('A');

    root->left->right = constructNode('B');

    root->right->left = constructNode('C');

    root->right->right = constructNode('D');

    return root;

}

int main(void) {

    TreeNode\* expRoot = constructExp();

    int choice;

    do {

        printf("\nMenu:\n");

        printf("1. Print Prefix Expression. \n");

        printf("2. Print Infix Expression.\n");

        printf("3. Print Postfix Expression.\n");

        printf("4. Exit\n");

        printf("Choose an option: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Prefix Expression: ");

                preorder(expRoot);

                printf("\n");

                break;

            case 2:

                printf("Infix Expression: ");

                inorder(expRoot);

                printf("\n");

                break;

            case 3:

                printf("Postfix Expression: ");

                postorder(expRoot);

                printf("\n");

                break;

            case 4:

                printf("Exiting...\n");

                break;

            default:

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

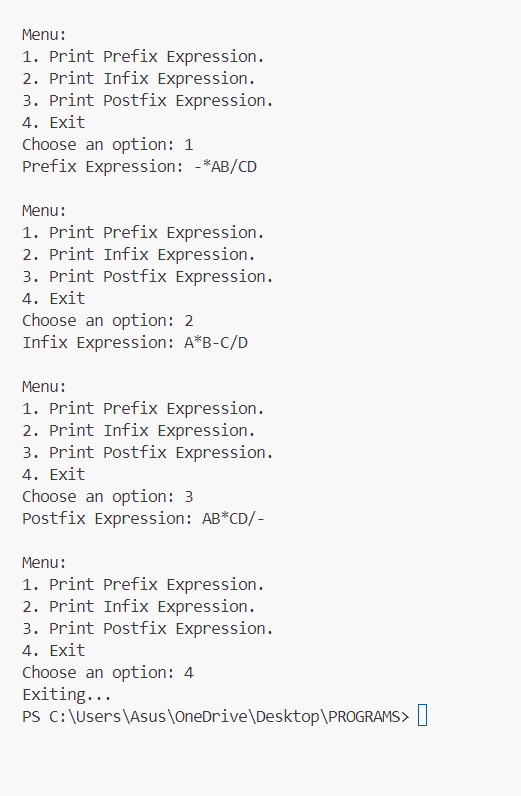
        }

    } while (choice != 4);

    return 0;

}

**OUTPUT:**

****