DELHI TECHNOLOGICAL UNIVERSITY

# (FORMELY DELHI COLLEGE OF ENGINEERING)

# Shahbad Daulatpur,Bawana Road,Delhi-110042

# DEPARTMENT OF SOFTWARE ENGINEERING



**SE-203**

**DATA STRUCTURES**

**LAB FILE**

SUBMITTED TO: SUBMITTED BY:

Mr. Ankur Narwal Aradhay jain(23/SE/30)

Department Of Software Engineering

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Objective** | **Date** | **Sign** |
| 1. | To reverse an array of characters | 21/08/24 |  |
| 2. | To perform various operations on array:   * Insert element * Delete element * Find largest element * Find smallest element | 04/09/24 |  |
| 3. | To perform string operations:   * Merge 2 strings * Reverse a string * Find and replace substring | 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. | 16/10/24 |  |
| 7. | Choose a unique expression and store it in a binary tree. Use appropriate tree traversal to generate postfix, prefix and infix. | 16/10/24 |  |

**EXPERIMENT-1**

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

**CODE**:

#include<stdio.h>

#include<stdlib.h>

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

int i=0;

int j=n-1;

while(j>i){

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

i++;

j--;

}

}

int main(){

int n;

printf("input size:");

scanf("%d",&n);

int\* arr=(int\*)malloc(n\*sizeof(int));

printf("enter data:");

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

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

}

reverse(arr,n);

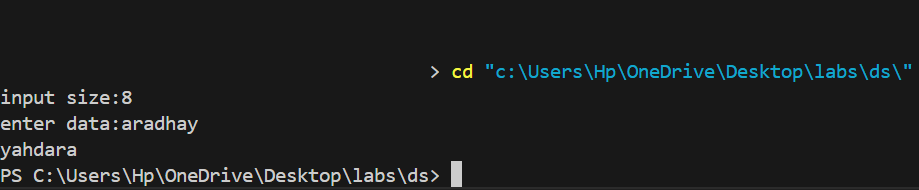
for(int i=0;i<n;i++){

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

}

}

OUTPUT:



**EXPERIMENT-2**

**AIM**: To perform various operations on array:

Insert element

Delete element

Find largest element

Find smallest element

**CODE**:

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

void add(int \*arr,int index,int value,int size){

if(size==0 && index==0){

arr[index]=value;

}

else{

for(int i=size-1;i>=index;i--){

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

}

arr[index]=value;

}

}

void deleteFromArray(int\* arr,int index,int size){

if(size==0){

printf("no value to delete \n");

}

if(index==size-1){

arr[index]=-1;

}

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

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

}

}

int findMax(int \*arr,int size){

int maxi=-1e9;

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

if(arr[i]>maxi){

maxi=arr[i];

}

}

return maxi;

}

int findMin(int \*arr,int size){

int mini=1e9;

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

if(arr[i]<mini){

mini=arr[i];

}

}

return mini;

}

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

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

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

}

printf("size is:%d",size);

}

int main(){

char ch;

int arr[100];

int size=0;

while(1){

int choice;

printf("1. add element\n");

printf("2. delete element\n");

printf("3. find maximum\n");

printf("4. find minimum\n");

printf("enter your choice:");

scanf("%d",&choice);

if(choice==1){

printf("enter value to add:");

int value;

scanf("%d",&value);

printf("enter index:");

int index;

scanf("%d",&index);

add(arr,index,value,size);

size++;

}

else if(choice==2){

int index;

printf("enter index to delete:");

scanf("%d",&index);

deleteFromArray(arr,index,size);

size--;

}

else if(choice==3){

printf("maximum element is %d \n",findMax(arr,size));

}

else if(choice==4){

printf("mainimum element is %d \n",findMin(arr,size));

}

else{

display(arr,size);

}

printf("to continue press y:");

getchar();

scanf("%c",&ch);

if(ch!='y'){

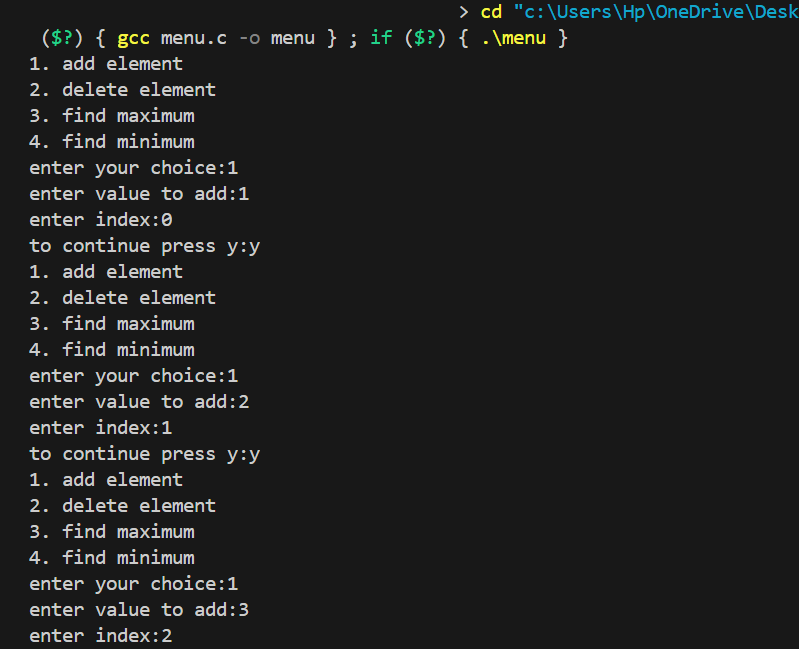
break;

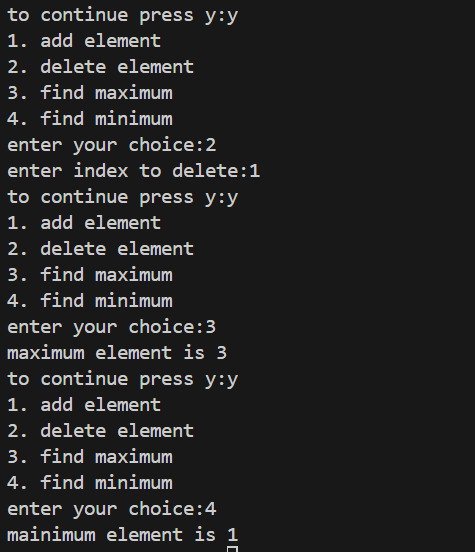
}

}

}

**OUTPUT:**





**EXPERIMENT-3**

**AIM**: To perform string operations:

Merge 2 strings

Reverse a string

Find and replace substring

**CODE**:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

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

    int length = strlen(str1) + strlen(str2) + 1;

    char\* str3 = (char\*)malloc(length \* sizeof(char));

    int k = 0;

    for (int i = 0; i < strlen(str1); i++) {

        str3[k++] = str1[i];

    }

    for (int i = 0; i < strlen(str2); i++) {

        str3[k++] = str2[i];

    }

    str3[k] = '\0';

    return str3;

}

void reverse(char\* str1){

    int i=0,j=strlen(str1)-1;

    while(j>i){

        char temp=str1[i];

        str1[i]=str1[j];

        str1[j]=temp;

        i++;

        j--;

    }

}

char\* substring(char\* str,char\* substr,char\* replacestr){

    int j=0,index,size=strlen(substr),flag1=0,flag2=0;

    for(int i=0;str[i]!='\0';i++){

        if(str[i]==substr[j]){

            flag1=1;

            int J=j;

            int k=i;

            while(size-1>0){

                if(str[k]!=substr[J]){

                    flag2=1;

                    break;

                }

                J++;

                size--;

                k++;

            }

            index=i;

            break;

        }

    }

    if(flag1=1 && flag2==0){

        int g=0;

        while(substr[g]!='\0'){

            str[index]=replacestr[g];

            index++;

            g++;

        }

         return str;

    }

    else{

        return "No such substring";

    }

}

int main() {

    int choice;

    char str1[50],str2[50],str3[50],str4[50],substr[30],newstr[30];

    while(1){

        printf("1. merge strings\n");

        printf("2. reverse string\n");

        printf("3. replace subtring in a string\n");

        printf("Enter your choice:");

        scanf("%d", &choice);

        if (choice == 1) {

            char str1[30], str2[10];

            printf("Enter the first string: ");

            getchar();

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

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

            printf("Enter the second string: ");

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

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

            char\* str3 = merge(str1, str2);

            printf("Merged string: ");

            puts(str3);

            free(str3);

        }

        else if(choice==2){

            int n;

            printf("enter length:");

            scanf("%d",&n);

            char\* str1 = (char\*)malloc(n \* sizeof(char));

            printf("enter word:");

            getchar();

            gets(str1);

            reverse(str1);

            puts(str1);

            free(str1);

        }

        else if(choice==3){

                printf("Enter main string : ");

                getchar();

                gets(str4);

                printf("Enter substring : ");

                gets(substr);

                printf("Enter new substring : ");

                gets(newstr);

                puts(substring(str4,substr,newstr));

        }

        else{

            printf("invalid choice");

        }

        printf("want to continue ?");

        char c;

        scanf("%c",&c);

        if(c!='y'){

            break;

        }else{

            continue;

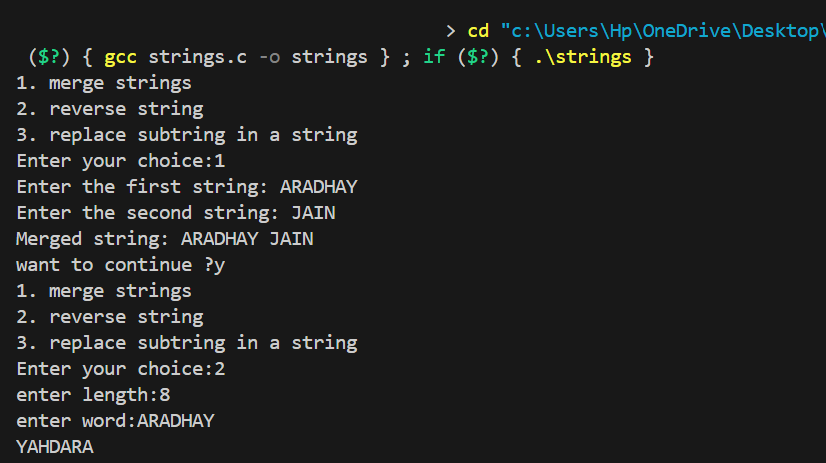
        }

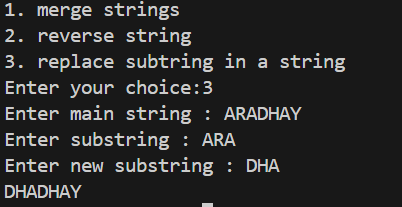
    }

    return 0;

}

**OUTPUT**:





**EXPERIMENT 4**

**AIM**: Write a program to implement character stack using an array  
Push  
Popup function using boundry condition  
Also write paranthesis correctness in a string array

**CODE**:

#include<stdio.h>

#include<stdlib.h>

typedef struct{

int top;

char\* arr;

int capacity;

}Stack;

int isEmpty(Stack\* s){

if(s->top==-1){

return 1;

}

return 0;

}

void initStack(Stack\* s,int n){

s->top=-1;

s->arr=(char\*)malloc(n\*sizeof(char));

s->capacity=n;

}

void pushStack(Stack\* s,char x){

if(s->top==s->capacity-1){

printf("overflow!\n");

return;

}

s->top++;

s->arr[s->top]=x;

}

char popStack(Stack\* s){

if(isEmpty(s)){

printf("underflow!\n");

return -1;

}

int ans=s->arr[s->top];

s->top--;

return ans;

}

char topStack(Stack\* s){

if(isEmpty(s)){

printf("underflow!\n");

return -1;

}

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

}

int matchingPair(char open, char close) {

return (open == '(' && close == ')') ||

(open == '{' && close == '}') ||

(open == '[' && close == ']');

}

int paranthesisChecker(Stack\* s, char\* str, int n) {

if (n % 2 != 0) {

return 0;

}

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

char elem = str[i];

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

pushStack(s, elem);

}

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

if (isEmpty(s)) {

return 0;

}

char topElem = popStack(s);

if (!matchingPair(topElem, elem)) {

return 0;

}

}

}

return isEmpty(s);

}

int main(){

Stack\* s1 = (Stack\*)malloc(sizeof(Stack));

int n1;

printf("Enter size of stack: ");

scanf("%d", &n1);

initStack(s1, n1);

char ch;

char ch1;

for (int i = 0; i < n1; i++) {

printf("Enter character: ");

getchar();

scanf("%c", &ch);

pushStack(s1, ch);

printf("If you want to add more, press 'y', else 'n': ");

getchar();

scanf("%c", &ch1);

if (ch1 == 'y') {

continue;

} else {

break;

}

}

printf("top of the stack is %c\n",topStack(s1));

free(s1->arr);

Stack\* s;

s = (Stack\*)malloc(sizeof(Stack));

char\* str;

int n;

printf("enter size of string:");

scanf("%d",&n);

str=(char\*)malloc((n+1)\*sizeof(char));

printf("enter string to check:");

getchar();

gets(str);

initStack(s,n);

if (paranthesisChecker(s, str, n)) {

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

} else {

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

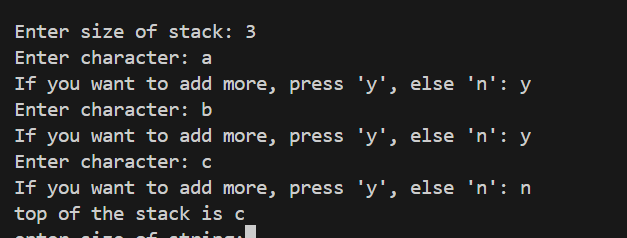
}

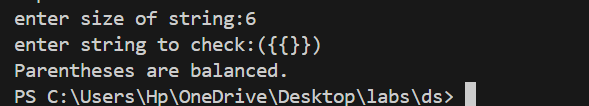
free(s->arr);

free(str);

}

**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 <stdlib.h>

typedef struct {

int front, rear, size;

int\* arr;

int capacity;

} CircularQueue;

void initQueue(CircularQueue\* q, int capacity) {

q->capacity = capacity;

q->front = -1;

q->rear = -1;

q->size = 0;

q->arr = (int\*)malloc(q->capacity \* sizeof(int));

}

int isFull(CircularQueue\* q) {

return (q->size == q->capacity);

}

int isEmpty(CircularQueue\* q) {

return (q->size == 0);

}

void enqueue(CircularQueue\* q, int value) {

if (isFull(q)) {

printf("Queue is full!\n");

return;

}

if (q->front == -1) {

q->front = 0;

}

q->rear = (q->rear + 1) % q->capacity;

q->arr[q->rear] = value;

q->size++;

printf("%d enqueued to queue\n", value);

}

int dequeue(CircularQueue\* q) {

if (isEmpty(q)) {

printf("Queue is empty!\n");

return -1;

}

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

q->front = (q->front + 1) % q->capacity;

q->size--;

if (q->size == 0) {

q->front = -1;

q->rear = -1;

}

printf("%d dequeued from queue\n", value);

return value;

}

void removeDuplicates(CircularQueue\* q) {

if (isEmpty(q)) {

printf("Queue is empty!\n");

return;

}

int\* seen = (int\*)calloc(1000, sizeof(int));

CircularQueue\* newQueue = (CircularQueue\*)malloc(sizeof(CircularQueue));

initQueue(newQueue, q->capacity);

int count = q->size;

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

int elem = dequeue(q);

if (!seen[elem]) {

enqueue(newQueue, elem);

seen[elem] = 1;

}

}

free(q->arr);

q->arr = newQueue->arr;

q->front = newQueue->front;

q->rear = newQueue->rear;

q->size = newQueue->size;

q->capacity = newQueue->capacity;

free(newQueue);

free(seen);

}

void display(CircularQueue\* q){

int size=q->size;

int j=q->front;

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

printf("%d\n",q->arr[j]);

j=(j+1)%q->capacity;

}

}

int main() {

CircularQueue\* q=(CircularQueue\*)malloc(sizeof(CircularQueue));

int capacity;

printf(“enter capacity:”);

scanf(“%d”,&capacity);

initQueue(q, capacity);

while(1){

int choice;

printf("enter your choice:\n");

printf("1. display queue\n");

printf("2. insert in queue\n");

printf("3. delete from queue\n");

printf("4. remove duplicates\n");

scanf("%d",&choice);

if(choice==1){

display(q);

}

else if(choice==2){

int x;

printf("enter element:");

scanf("%d",&x);

enqueue(q,x);

}

else if(choice==3){

dequeue(q);

}

else if(choice==4){

removeDuplicates(q);

}

char ch;

printf("want to continue:");

getchar();

scanf("%c",&ch);

if(ch!='y'){

break;

}

else{

continue;

}

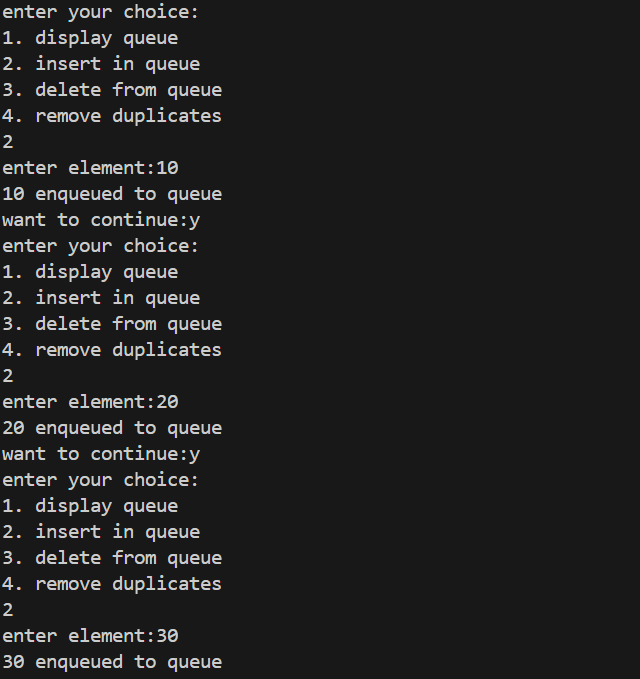
}

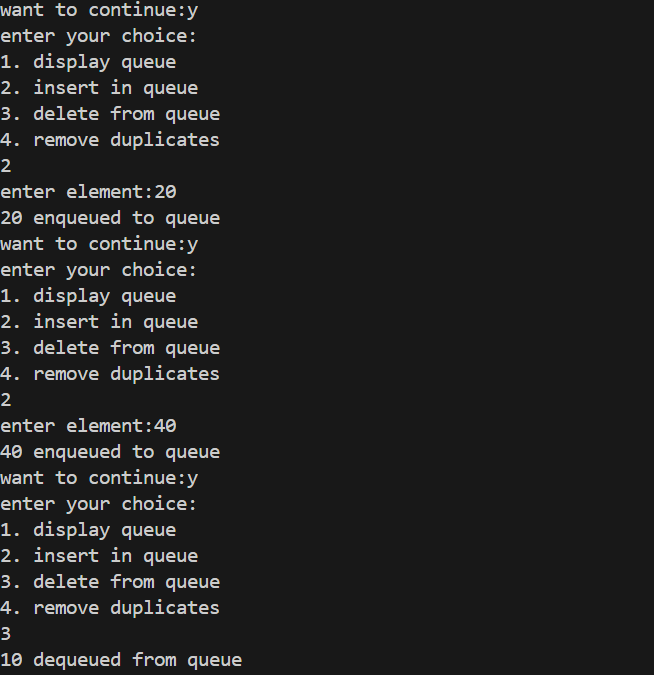
free(q->arr);

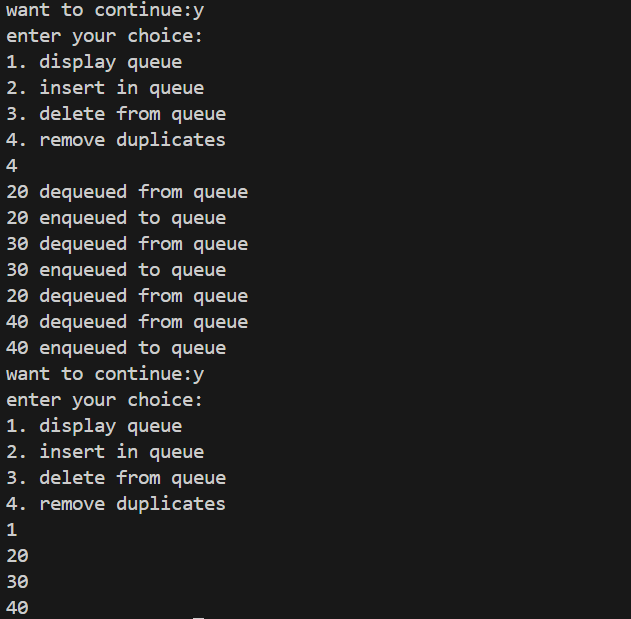
return 0;

}

**OUTPUT:**







**EXPERIMENT 6**

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

**CODE:**

#include<stdio.h>

#include<stdlib.h>

//Aradhay jain 23-SE-30

typedef struct Node {

    int data;

    struct Node\* prev;

    struct Node\* next;

} Node;

Node\* createNode(int x) {

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

    newElement->data = x;

    newElement->prev = NULL;

    newElement->next = NULL;

    return newElement;

}

void display(Node\* head) {

    Node\* temp = head;

    if (temp == NULL) {

        printf("List is empty.\n");

        return;

    }

    printf("Doubly Linked List: ");

    while (temp != NULL) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

    printf("\n");

}

Node\* insertNode(int x, Node\* head, int position) {

    Node\* newElement = createNode(x);

    if (position == 1) {

        newElement->next = head;

        if (head != NULL) {

            head->prev = newElement;

        }

        head = newElement;

        return head;

    }

    Node\* temp = head;

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

        temp = temp->next;

    }

    if (temp == NULL) {

        free(newElement);

        return head;

    }

    newElement->next = temp->next;

    newElement->prev = temp;

    if (temp->next != NULL) {

        temp->next->prev = newElement;

    }

    temp->next = newElement;

    return head;

}

Node\* deleteNode(Node\* head, int position) {

    if (head == NULL) return head;

    Node\* temp = head;

    if (position == 1) {

        head = temp->next;

        if (head != NULL) {

            head->prev = NULL;

        }

        free(temp);

        return head;

    }

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

        temp = temp->next;

    }

    if (temp == NULL) return head;

    if (temp->prev != NULL) {

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

    }

    if (temp->next != NULL) {

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

    }

    free(temp);

    return head;

}

int main() {

    Node\* head = NULL;

    int choice, data, size;

    while (1) {

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

        printf("1. Display list\n");

        printf("2. Insert data\n");

        printf("3. Delete data\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        if (choice == 1) {

            display(head);

        } else if (choice == 2) {

            int value;

            printf("enter value to add: ");

            scanf("%d", &value);

            int pos;

            printf("enter position: ");

            scanf("%d", &pos);

            head = insertNode(value, head, pos);

        } else if (choice == 3) {

            int pos;

            printf("enter position: ");

            scanf("%d", &pos);

            head = deleteNode(head, pos);

        } else if (choice == 4) {

            break;

        } else {

            printf("invalid choice\n");

        }

        char ch;

        printf("to continue press y: ");

        getchar();

        scanf("%c", &ch);

        if (ch != 'y') {

            break;

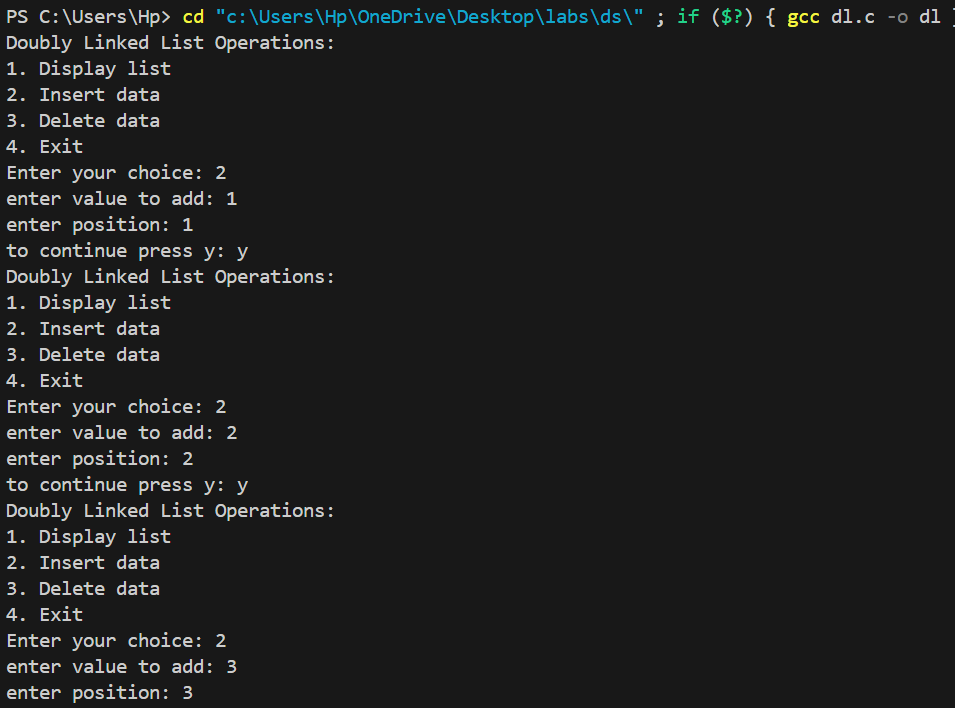
        }

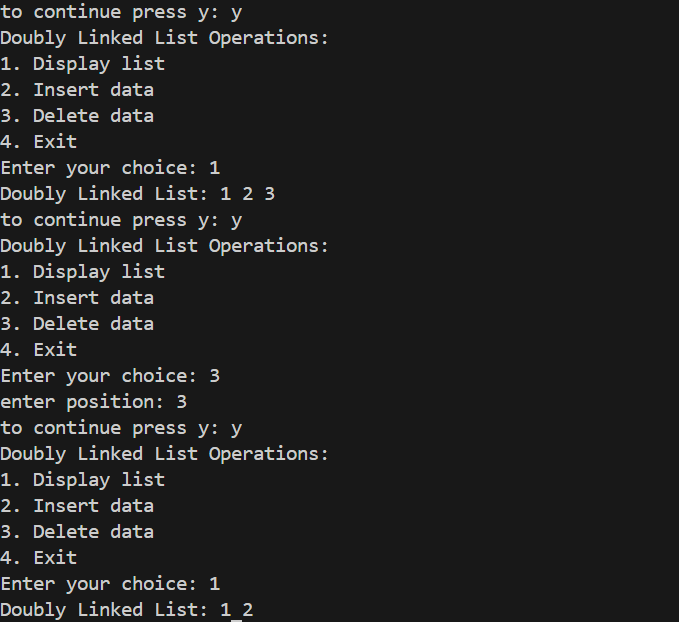
    }

    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>

//Aradhay jain 23-SE-30

typedef struct Node {

    char value;

    struct Node\* left;

    struct Node\* right;

}Node;

Node\* createNode(char value) {

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

    newNode->value = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

void preOrder(Node\* root) {

    if (root == NULL) return;

    printf("%c ", root->value);

    preOrder(root->left);

    preOrder(root->right);

}

void postOrder(Node\* root) {

    if (root == NULL) return;

    postOrder(root->left);

    postOrder(root->right);

    printf("%c ", root->value);

}

void inorder(Node\* root){

    if(root==NULL) return;

    inorder(root->left);

    printf("%c ",root->value);

    inorder(root->right);

}

int main(){

    Node\* root=createNode('/');

    root->left=createNode('+');

    root->right=createNode('-');

    root->left->left=createNode('A');

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

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

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

    while(1){

        int choice;

        printf("options available:\n");

        printf("1. generate postfix expression\n");

        printf("2. generate prefix expression\n");

        printf("3. generate infix expression\n");

        printf("enter your choice:");

        scanf("%d",&choice);

        if(choice==1){

            postOrder(root);

            printf("\n");

        }

        else if(choice==2){

            preOrder(root);

            printf("\n");

        }

        else if(choice==3){

            inorder(root);

            printf("\n");

        }

        else{

            printf("invalid choice");

        }

        printf("to continue press y:");

        char ch;

        getchar();

        scanf("%c",&ch);

        if(ch!='y'){

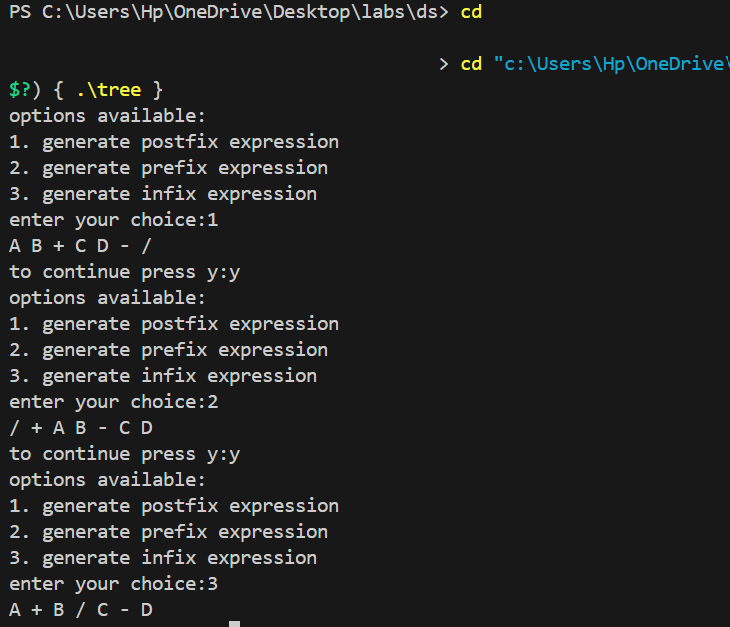
            break;

        }

    }

}

**OUTPUT:**

****