1. **Write a C Program to demonstrate Stack operations using arrays.**

#include<stdio.h>

#define Max 5

int Push(int[],int);

int Pop(int[],int);

void Show(int[],int);

int main(){

int ch,stack[Max],top=-1;

for(;;)

{

printf("1.Push\n2.Pop\n3.Show\n4.Exit\nEnter your choice");

scanf("%d",&ch);

switch(ch)

{

case 1: top=Push(stack,top);

break;

case 2: top=Pop(stack,top);

break;

case 3: Show(stack,top);

break;

case 4: return 0;

default:printf("Invalid choice\nPlease enter valid choice\n");

}//enf of switch

}//end of for

}//end of main

//Push operation

int Push(int s[Max], int t)

{

int element;

if(t==Max-1)

printf("Stack is full....!\n");

else

{

printf("Enter the elements");

scanf("%d",&element);

t+=1;

s[t]=element;

}

return t;

}

//Pop Operation

int Pop(int s[Max],int Top)

{

int element;

if(Top==-1)

printf("Stack is empty");

else

{

element=s[Top];

printf("%d is popped",element);

Top-=1;

}

return(Top);

}

//Display

void Show(int s[Max],int T)

{

int i;

if(T==-1)

printf("stack is empty");

else

{

for(i=0;i<=T;i++)

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

}

}

**2. Write a C Program to evaluate postfix expression, postfix expression contains single digit integers and the operators +,-,\*and /.**

#include <stdio.h>

#define max 10

int push(int[], int, int);

int pop(int[], int\*);

int evaluatePostfix(char postfix[]) {

int stack[max];

int top = -1; // Initialize stack top

int i = 0, op1, op2, result;

char symb;

while (postfix[i] != '\0') {

symb = postfix[i];

if (symb >= '0' && symb <= '9') {

top = push(stack, top, symb - '0'); // Push converted integer

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

op2 = pop(stack, &top); // Get second operand

op1 = pop(stack, &top); // Get first operand

// Perform the operation

if (symb == '+')

result = op1 + op2;

else if (symb == '-')

result = op1 - op2;

else if (symb == '\*')

result = op1 \* op2;

else if (symb == '/')

result = op1 / op2;

top = push(stack, top, result); // Push the result onto the stack

}

i++;

}

return pop(stack, &top); // Final result

}

int push(int s[max], int t, int ele) {

if (t == max - 1) {

printf("Stack is full\n");

} else {

s[++t] = ele; // Increment top and push element

}

return t;

}

int pop(int s[max], int \*top) {

if (\*top == -1) {

printf("Stack is empty\n");

return -1; // Indicate error

} else {

return s[(\*top)--]; // Return element and decrement top

}

}

int main() {

char postfix[25];

printf("Enter the postfix expression: ");

scanf("%s", postfix);

int result = evaluatePostfix(postfix);

printf("The result of postfix expression %s is: %d\n", postfix, result);

return 0;

}

//output:Enter the postfix expression: 23+5\*

//The result of postfix expression 23+5\* is: 25

//------------------

//(program exited with code: 0)

//Press return to continue

**3.Write a C Program to convert infix to postfix expression**.

#include <stdio.h>

#include <string.h>

#define MAX 100

// Function to push an element onto the stack

int push(int s[MAX], int t, int ele) {

if (t == MAX - 1) {

printf("Stack is full\n");

} else {

s[++t] = ele; // Increment top and push element

}

return t;

}

// Function to pop an element from the stack

int pop(int s[MAX], int \*top) {

if (\*top == -1) {

printf("Stack is empty\n");

return -1; // Indicate error

} else {

return s[(\*top)--]; // Return element and decrement top

}

}

// Function to check precedence of operators

int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

// Function to convert infix to postfix

void infixToPostfix(char\* infix, char\* postfix) {

int stack[MAX];

int top = -1; // Initialize stack top

int k = 0;

for (int i = 0; infix[i]; i++) {

char symb = infix[i];

// Directly check if the symbol is an operand (A-Z or 0-9)

if ((symb >= 'A' && symb <= 'Z') || (symb >= '0' && symb <= '9')) {

postfix[k++] = symb;

}

// If the symbol is '(', push it to the stack

else if (symb == '(') {

top = push(stack, top, symb);

}

// If the symbol is ')', pop until '(' is found

else if (symb == ')') {

while (top != -1 && stack[top] != '(') {

postfix[k++] = pop(stack, &top);

}

pop(stack, &top); // Pop '('

}

// If the symbol is an operator

else {

while (top != -1 && precedence(stack[top]) >= precedence(symb)) {

postfix[k++] = pop(stack, &top);

}

top = push(stack, top, symb);

}

}

// Pop all remaining operators from the stack

while (top != -1) {

postfix[k++] = pop(stack, &top);

}

postfix[k] = '\0'; // Null-terminate the postfix expression

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter an infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

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

return 0;

}

1. **Write a C Program to demonstrate Queue operations using arrays.**

#include <stdio.h>

#define Max 5 // Define the maximum size of the queue

// Function to insert an element into the queue

int Insert(int q[Max], int rear) {

int ele;

if (rear == Max - 1) {

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

} else {

printf("Enter the element to be inserted: ");

scanf("%d", &ele);

rear++;

q[rear] = ele;

}

return rear;

}

// Function to delete an element from the queue

int Delete(int q[Max], int front, int rear) {

int ele;

if (front > rear) {

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

} else {

ele = q[front];

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

front++;

}

return front;

}

// Function to display the elements of the queue

void Show(int q[Max], int front, int rear) {

if (front > rear) {

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

} else {

printf("Queue elements: ");

for (int i = front; i <= rear; i++) { // Loop from front to rear

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

}

printf("\n");

}

}

int main() {

int queue[Max];

int choice;

int front = 0, rear = -1;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Insert\n2. Delete\n3. Show\n4. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

rear = Insert(queue, rear);

break;

case 2:

front = Delete(queue, front, rear);

break;

case 3:

Show(queue, front, rear);

break;

case 4: // Add exit option

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

return 0;

default:

printf("Invalid queue choice!\n");

break;

}

}

return 0;

}

1. **Write a C Program to demonstrate different operations on singly linked list**.

#include<stdio.h>

#include<stdlib.h>

struct Node{

int info;

struct Node\* link;

};

typedef struct Node\* NODE;

//Function declarations

NODE Insert\_Front(int,NODE);

NODE Insert\_End(int,NODE);

NODE Delete\_Front(NODE);

NODE Delete\_End(NODE);

void display(NODE);

//Function to insert the element at front of linked list

NODE Insert\_Front(int e, NODE First){

NODE New = (NODE)malloc(sizeof(struct Node));

New->info=e;

New->link=NULL;

//chech if list empty and newly created is first node

if(First==NULL){

return New;

}

New->link=First;

printf("Insert successfully\n");

return New;

}

//Function to insert the element at End of linked list

NODE Insert\_End(int e, NODE First){

NODE temp=First,New;

New = (NODE)malloc(sizeof(struct Node));

New->info=e;

New->link=NULL;

//chech if list empty and newly created is first node

if(First==NULL){

return New;

}

while(temp->link!=NULL){

temp=temp->link;

}

temp->link=New;

printf("Insert successfully\n");

return First;

}

// Function to delete from the front

NODE Delete\_Front(NODE First) {

// If the list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// If the list has one item

if (First->link == NULL) {

printf("%d deleted element\n", First->info);

free(First);

return NULL; // The list will now be empty

}

NODE temp = First;

// More than one item in the list

First = First->link; // Move head to the next node

printf("%d deleted element\n", temp->info);

free(temp); // Free the old head

return First; // Return the new head of the list

}

// Function to delete from the front rear

NODE Delete\_End(NODE First){

// If the list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// If the list has one item

if (First->link == NULL) {

printf("%d deleted element\n", First->info);

free(First);

return NULL;

}

NODE Next=First,prev=NULL;

while(Next->link!=NULL){

prev=Next;

Next=Next->link;

}

prev->link=NULL;

printf("%d deleted element\n", Next->info);

free(Next);

return First;

}

void display(NODE First) {

NODE temp = First;

// If the list is empty

if (First == NULL) {

printf("List is empty\n");

return;

}

printf("\n");

printf("Linked List: \n");

while (temp != NULL) {

printf("%d\t", temp->info);

temp = temp->link;

}

printf("\n");

}

// Main function

int main() {

NODE First = NULL;

int choice, ele;

while (1) {

printf("1. Insert at Front\n2. Insert at End\n3. Delete from Front\n4. Delete from End\n5. Display List\n6. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at front: ");

scanf("%d", &ele);

First = Insert\_Front(ele, First);

break;

case 2:

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

scanf("%d", &ele);

First = Insert\_End(ele, First);

break;

case 3:

First = Delete\_Front(First);

break;

case 4:

First = Delete\_End(First);

break;

case 5:

display(First);

break;

case 6:

exit(0);

default:

printf("Invalid choice, please try again.\n");

}

}

return 0;

}

**6. Write a C Program to demonstrate different operations on circular doubly linked list.**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int info;

struct Node\* prev;

struct Node\* next;

};

typedef struct Node\* NODE;

//Function declarations

NODE Insert\_Front(int, NODE);

NODE Insert\_End(int, NODE);

NODE Delete\_Front(NODE);

NODE Delete\_End(NODE);

void display(NODE);

//Function to insert the element at the front of the linked list

NODE Insert\_Front(int e, NODE First){

NODE New, temp;

New = (NODE)malloc(sizeof(struct Node));

New->info = e;

New->prev = NULL;

New->next = NULL;

// Check if list is empty

if (First == NULL) {

New->next = New;

New->prev = New;

return New;

}

// Non-empty list, find the last node

temp = First;

while (temp->next != First) {

temp = temp->next;

}

// Insert New node at the front

New->next = First;

New->prev = temp;

First->prev = New;

temp->next = New;

printf("Inserted %d at the front successfully\n", e);

return New;

}

// Function to insert element at end of linked list

NODE Insert\_End(int e, NODE First){

NODE New, temp;

New = (NODE)malloc(sizeof(struct Node));

New->info = e;

New->prev = NULL;

New->next = NULL;

// Check if list is empty

if (First == NULL) {

New->next = New;

New->prev = New;

return New;

}

// Find last node

temp = First;

while (temp->next != First) {

temp = temp->next;

}

// Insert new node at the end

New->prev = temp;

New->next = First;

temp->next = New;

First->prev = New;

printf("Inserted %d at the end successfully\n", e);

return First;

}

// Function to delete the front node

NODE Delete\_Front(NODE First) {

NODE temp = First, temp1 = First;

// Check if list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// Check if there is only one node

if (First->next == First) {

printf("%d deleted\n", First->info);

free(First);

return NULL;

}

// Find the last node

while (temp->next != First) {

temp = temp->next;

}

First = First->next;

printf("%d deleted\n", temp1->info); // Corrected to print the front node's info

First->prev = temp;

temp->next = First;

free(temp1);

return First;

}

// Function to delete the last node

NODE Delete\_End(NODE First) {

NODE temp = First, temp1 = NULL;

// Check if list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// Check if there is only one node

if (First->next == First) {

printf("%d deleted\n", First->info);

free(First);

return NULL;

}

// Find the last node

while (temp->next != First) {

temp = temp->next;

}

temp1 = temp->prev;

printf("%d deleted\n", temp->info);

temp1->next = First;

First->prev = temp1;

free(temp);

return First;

}

// Function to display the circular doubly linked list

void display(NODE First) {

NODE temp;

if (First == NULL) {

printf("List is empty\n");

return;

}

temp = First;

do {

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

temp = temp->next;

} while (temp != First);

printf("\n");

}

// Main function

int main() {

NODE First = NULL;

int choice, ele;

while (1) {

printf("1. Insert at Front\n2. Insert at End\n3. Delete from Front\n4. Delete from End\n5. Display List\n6. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at front: ");

scanf("%d", &ele);

First = Insert\_Front(ele, First);

break;

case 2:

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

scanf("%d", &ele);

First = Insert\_End(ele, First);

break;

case 3:

First = Delete\_Front(First);

break;

case 4:

First = Delete\_End(First);

break;

case 5:

display(First);

break;

case 6:

exit(0);

default:

printf("Invalid choice, please try again.\n");

}

}

return 0;

}

**7.Write a C program to implement the following operation on binary treeusing array:**

**i. Insert**

**ii. Delete**

**iii. Tree traversal**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int tree[MAX]; // Array to store the binary tree

int size = 0; // Number of elements in the tree

// Function to insert an element into the binary tree

void insert(int value) {

if (size == MAX) {

printf("Tree is full\n");

return;

}

tree[size] = value;

size++;

}

// Function to delete an element from the binary tree

void delete(int value) {

if (size == 0) {

printf("Tree is empty\n");

return;

}

int i;

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

if (tree[i] == value) {

break;

}

}

if (i == size) {

printf("Element not found in the tree\n");

return;

}

// Replace the element to be deleted with the last element

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

size--;

}

// In-order traversal of the binary tree

void inorder(int index) {

if (index >= size || tree[index] == -1) return;

inorder(2 \* index + 1); // Left child

printf("%d ", tree[index]);

inorder(2 \* index + 2); // Right child

}

// Pre-order traversal of the binary tree

void preorder(int index) {

if (index >= size || tree[index] == -1) return;

printf("%d ", tree[index]);

preorder(2 \* index + 1); // Left child

preorder(2 \* index + 2); // Right child

}

// Post-order traversal of the binary tree

void postorder(int index) {

if (index >= size || tree[index] == -1) return;

postorder(2 \* index + 1); // Left child

postorder(2 \* index + 2); // Right child

printf("%d ", tree[index]);

}

int main() {

int choice, value;

while (1) {

printf("\nBinary Tree Operations\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. In-order Traversal\n");

printf("4. Pre-order Traversal\n");

printf("5. Post-order Traversal\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &value);

insert(value);

break;

case 2:

printf("Enter the value to delete: ");

scanf("%d", &value);

delete(value);

break;

case 3:

printf("In-order Traversal: ");

inorder(0);

printf("\n");

break;

case 4:

printf("Pre-order Traversal: ");

preorder(0);

printf("\n");

break;

case 5:

printf("Post-order Traversal: ");

postorder(0);

printf("\n");

break;

case 6:

exit(0);

break;

default:

printf("Invalid choice\n");

}

}

return 0;

}