**1. Write a C program to implement a stack using an array with the following operations:**

**1. push()**

**2. pop()**

**3. peek()**

**4. display()**

**2. Repeat the above with menu driven approach.**

#include<stdio.h>

#include <stdlib.h>

#define max 10

//arr

int stack[max];

//top

int top =-1;

//handle the input or output error while pushing or poping

void handle\_error(const char \*message) {

printf("Error: %s\n", message);

exit(EXIT\_FAILURE); // Exit the program

}

//push

void push(int data){

//check if stack is full

if(top == max-1){

handle\_error("stack is full");

}

//

top++;

stack[top]=data;

}

//pop

void pop(){

//check if stack is empty

if(top == -1 ){

handle\_error("stack is empty");

}

stack[top--];

}

//peek

int peek(){

//check if stack is empty

if(top == -1 ){

handle\_error("stack is empty");

}

//return the top element

printf("Top element is =%d ",stack[top]);

return 0;

}

int display(){

//check if stack is empty

if(top == -1 ){

handle\_error("stack is empty");

}

int i;

//return all elements

printf("displaying the elments \n");

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

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

}

printf("\n");

return 0;

}

int main(){

int choice,data;

//menu driven

do{

printf("\n Enter the choice \n 1.Push \n 2.Pop \n 3.Peek \n 4. Dispplay \n 5.exit \n ::=");

scanf("%d",&choice);

switch(choice){

case 1:

printf("Enter the data to push=");

scanf("%d",&data);

push(data);

printf("Pushed == %d \n ",data);

break;

case 2:

pop();

printf("pop \n" );

break;

case 3:

printf("peek opp = %d \n",peek());

break;

case 4:

printf(" %d",display());

break;

case 5:

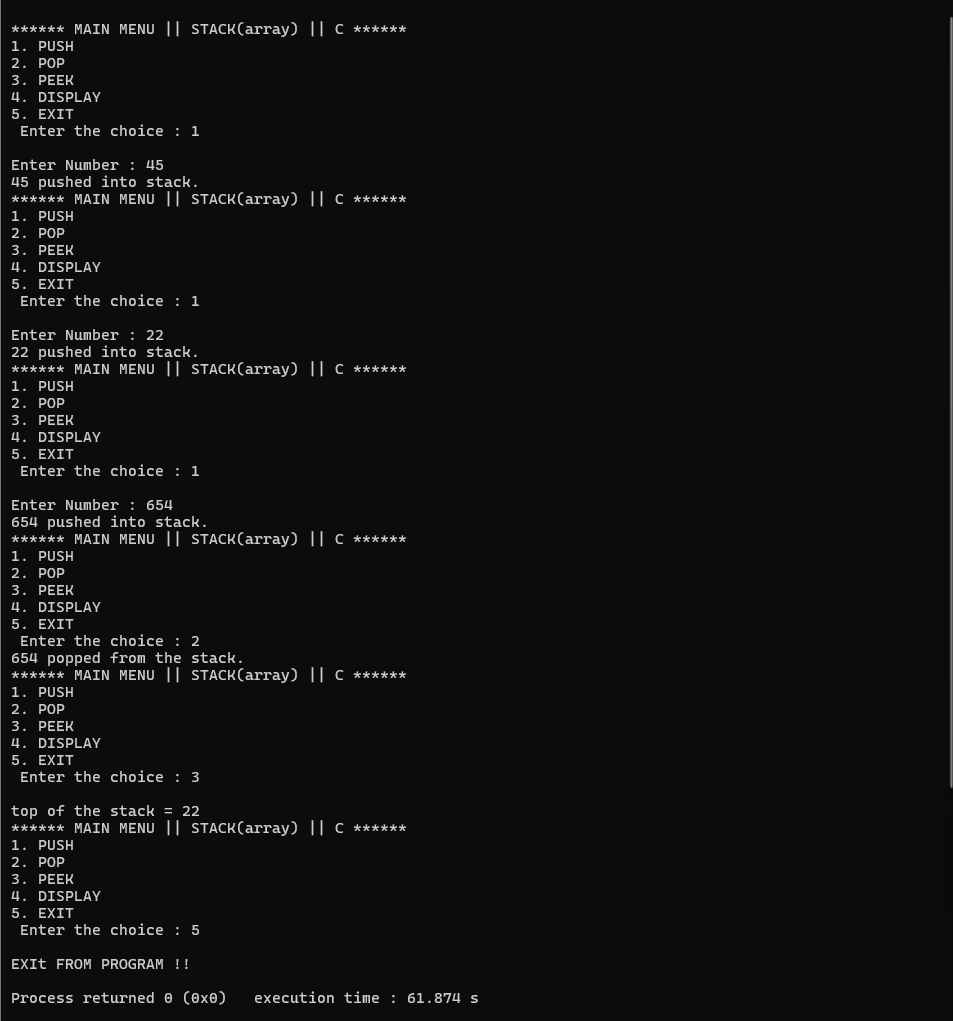
printf("exiting the menu");

break;

}}while(choice != 5);

return 0;

}



3. **Reimplement Q-1 in C++ using a class-based approach.**

//3. Reimplement Q-1 in C++ using a class-based approach.

//program to implement a stack using an array with the following operations:

//1. push()

//2. pop()

//3. peek()

//4. display()

#include<iostream>

using namespace std;

class Stack{

public:

//fuctions delcareation

void push(int );

void pop();

void peek();

void display();

private:

//delcareation of stack and top

static const int max =11;

int top =-1;

int stack[max];

};

void Stack::push(int data){

if (top == max) {

cout <<"The stack is Full ";

return;

}

top++;

stack[top]=data;

}

void Stack::pop(){

if (top == -1) {

cout<<"The stack is empty";

return;

}

top--;

}

void Stack::peek(){

if (top == -1) {

cout<<"The stack is empty";

return;

}

cout<<"The top is = "<<stack[top];

}

void Stack::display(){

if (top == -1) {

cout<<"The stack is empty";

return;

}

cout<<"The Stack is = "<<endl;

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

cout<<stack[i]<<"->";

}

}

int main(){

int choice,data;

Stack opp;

do {

cout<<"1.Push"<<endl;

cout<<"2.pop"<<endl;

cout<<"3.peek"<<endl;

cout<<"4.abort"<<endl;

cout<<"Enter the choice"<<endl;

cin>>choice;

switch (choice) {

case 1:

cout<<"Enter the data=";

cin>>data;

opp.push(data);

cout<<"Done"<<endl;

break;

case 2:

cout<<"POPING element";

opp.pop();

break;

case 3:

cout<<"peek";

opp.peek();

break;

case 4:

cout<<"exit";

break;

case 5:

cout<<"display ="<<endl;

opp.display();

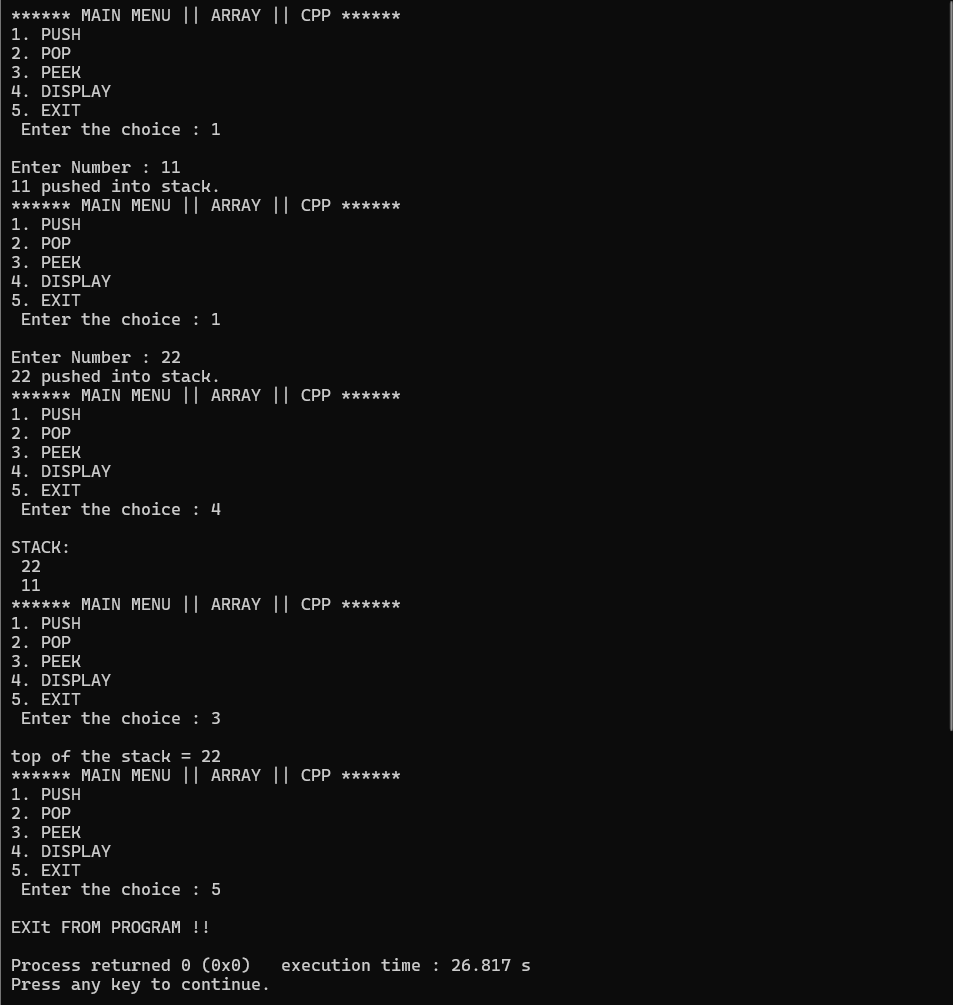
break;

}

} while(choice !=4);

return 0;

}

****

**4. Implement a stack using a singly linked list with push, pop, and display in C.**

**5. Repeat the above with menu driven approach.**

// 4. Implement a stack using a singly linked list with push, pop, and display

// in C.

// #include <algorithm>

// #include <cstdlib>

#include <stdio.h>

#include <stdlib.h>

// define struct

struct node {

int data;

struct node \*next;

} \*top = NULL;

// void push

int push(int);

// void pop

void pop();

// void display

void display();

// int main

int main() {

int choice, data;

// hard coded push pop display

// push(22);

// push(22);

// push(22);

// push(22);

// display();

// pop();

// pop();

do {

printf(

"Enter the options \n 1.push \n 2.pop \n 3 .display \n 4.abort === \n");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the data:=");

scanf("%d", &data);

push(data);

break;

case 2:

pop();

case 3:

display();

}

} while (choice != 4);

return 0;

}

int push(int data) {

// create a new node

struct node \*newnode;

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data = data;

if (top == NULL) {

newnode->next = NULL;

}

newnode->next = top;

top = newnode;

printf("insertation sucessful \n");

return 0;

}

void pop() {

if (top == NULL) {

printf("the stack is empty");

}

struct node \*temp = top;

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

top = temp->next;

free(temp);

}

void display() {

if (top == NULL) {

printf("the stack is empty \n");

}

struct node \*temp = top;

while (temp->next != NULL) {

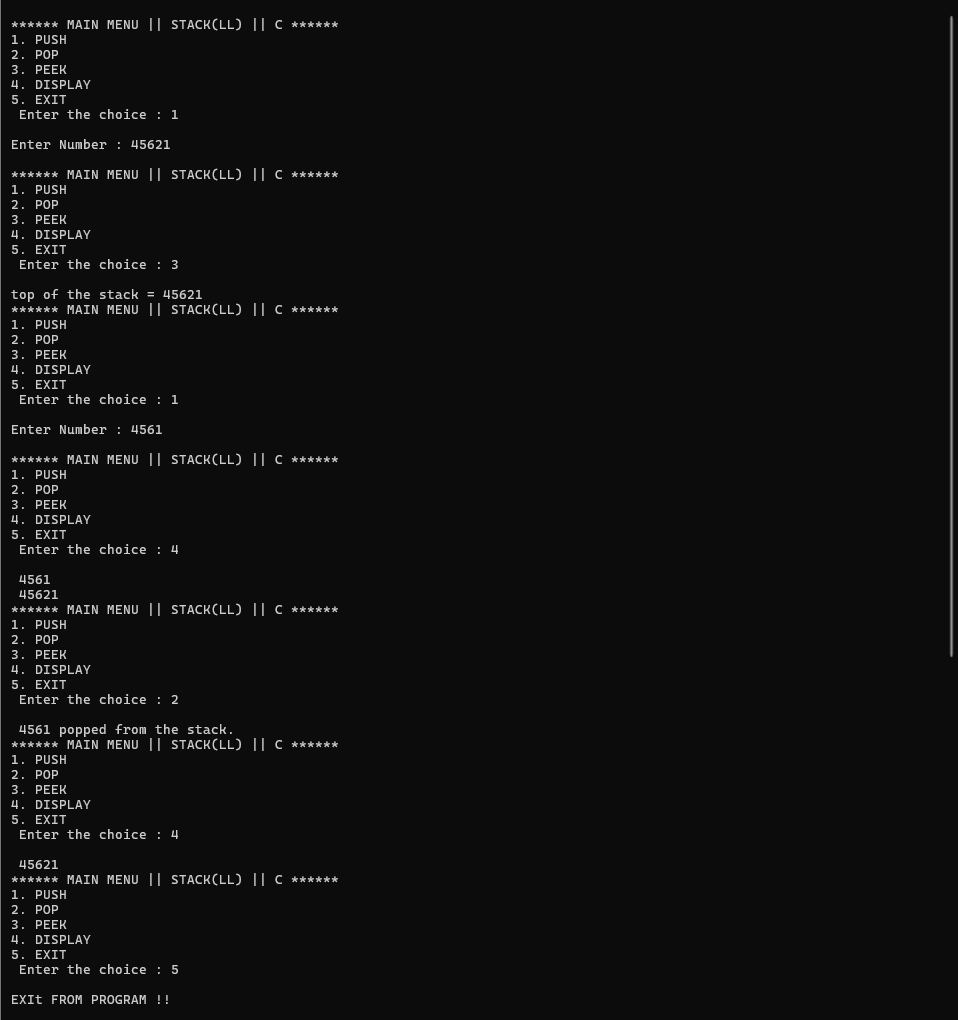
printf("%d->", temp->data);

temp = temp->next;

}

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

}

****

**6. Implement stack operations using Linked List in C++ with classes and objects.**

// Implement stack operations using Linked List in C++ with classes and objects.

#include <cstdlib>

#include <iostream>

#include <ostream>

using namespace std;

class node {

public:

node \*next;

int data;

node(int val) {

data = val;

next = nullptr;

}

};

class stack {

node \*top;

public:

stack() { top = nullptr; }

int push(int data);

void pop();

void display();

};

int stack::push(int data) {

node \*newnode = new node(data);

;

newnode->data = data;

if (top == NULL) {

newnode->next = NULL;

}

newnode->next = top;

top = newnode;

cout << "insertation sucessful \n" << endl;

return 0;

}

void stack::pop() {

if (top == NULL) {

cout << "the stack is empty" << endl;

return;

}

struct node \*temp = top;

cout << "deleted :" << temp->data << endl;

top = temp->next;

delete temp;

}

void stack::display() {

if (top == NULL) {

cout << "the stack is empty \n" << endl;

return;

}

struct node \*temp = top;

while (temp != NULL) {

cout << temp->data << "->" << endl;

temp = temp->next;

}

cout << "--->NULL" << endl;

}

int main() {

stack s;

s.push(12);

s.push(13);

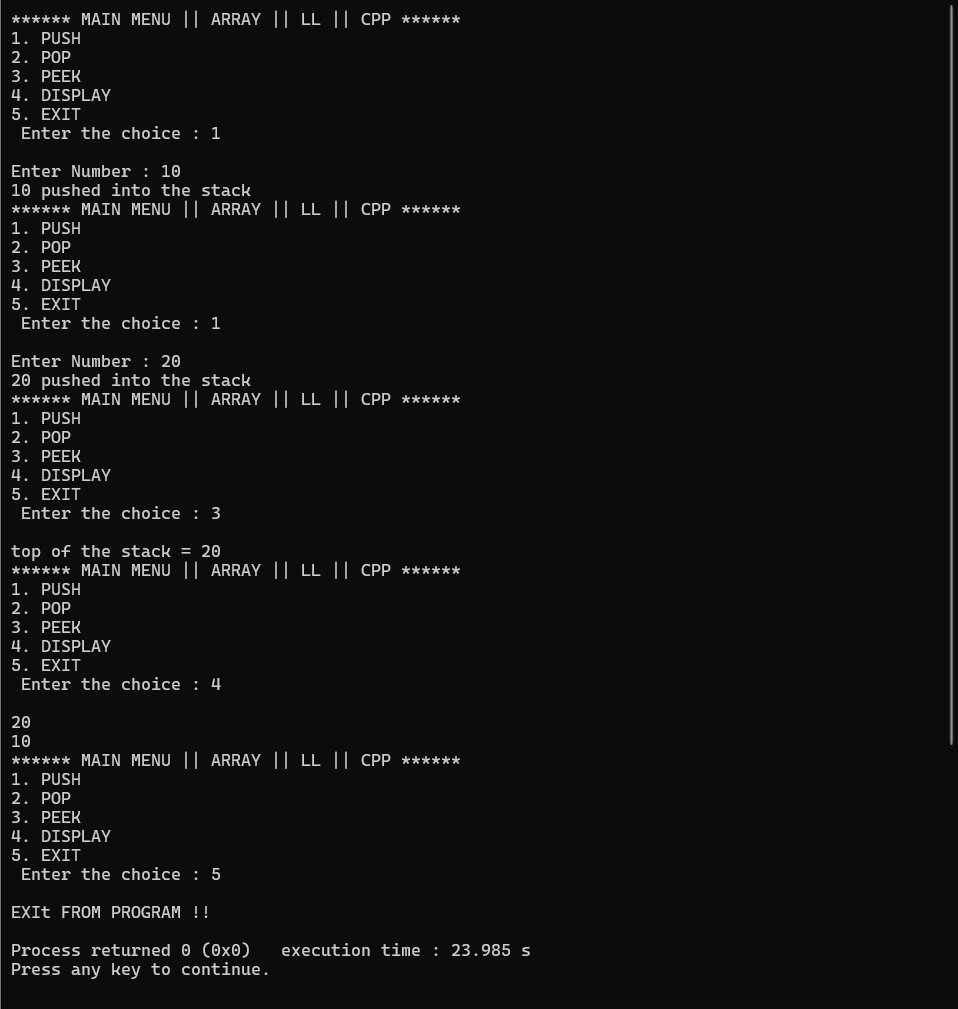
s.push(14);

s.push(15);

s.pop();

s.display();

}

****

**7. Write a program to demonstrate the use of the stack container in C++ STL by performing:**

**o Push elements**

**o Pop elements**

**o Access top() element**

**o Display size and empty status**

// Write a program to demonstrate the use of the stack container in C++ STL by

// performing:

// o Push elements

// o Pop elements

// o Access top() element

// o Display size and empty status

#include <iostream>

#include <stack>

using namespace std;

int main() {

stack<int> st;

if (st.empty()) {

cout << " At first Stack is empty " << endl;

}

if (!st.empty()) {

cout << "Stack is not empty. Top element: " << st.top() << endl;

}

st.push(10);

st.push(5);

cout << "Pushed 10->5" << endl;

// Accessing top element

cout << "Top element: " << st.top() << endl;

cout << "Size of stack:" << st.size() << endl;

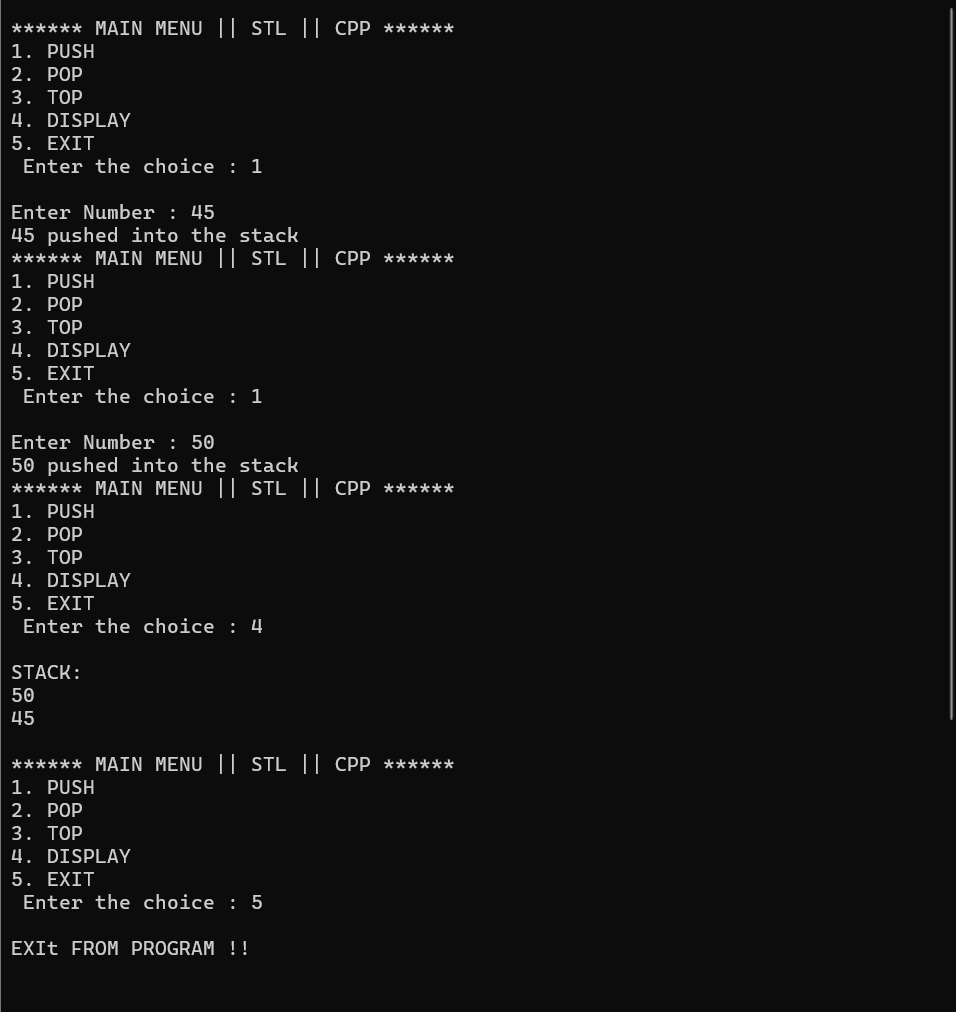
// Popping an element

st.pop();

cout << "Top element after pop: " << st.top() << endl;

return 0;

}



**8. Write a program in C to implement two stacks in one array such that:**

**o Stack 1 grows from left to right.**

**o Stack 2 grows from right to left.**

**o The program should support push1(), push2(), pop1(), pop2() and check for**

**overflow/underflow conditions.**

// 8. Write a program in C to implement two stacks in one array such that:

// o Stack 1 grows from left to right.

// o Stack 2 grows from right to left.

// o The program should support push1(), push2(), pop1(), pop2() and check for

// overflow/underflow conditions.

// Hint:

// • Use a single array arr[MAX].

// • Maintain two top variables:

// int top1 = -1;

// int top2 = MAX;

// • Stack1 elements go into arr[++top1].

// • Stack2 elements go into arr[--top2].

// • Overflow occurs if top1 + 1 == top2.

#include <stdio.h>

#define max 10

int stack[max], top1 = -1, top2 = max;

void push1(int val) {

if (top1 == top2 - 1) {

printf("\nSTACK OVERFLOW");

} else {

top1++;

stack[top1] = val;

printf("%d pushed into the stack", stack[top1]);

}

}

void pop1() {

if (top1 == -1) {

printf("\nSTACK UNDERFLOW");

} else {

printf("%d popped from the stack", stack[top1]);

top1--;

}

}

void display1() {

if (top1 == -1) {

printf("\nSTACK UNDERFLOW");

} else {

printf("stack:");

for (int i = top1; i >= 0; i--) {

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

}

}

}

void push2(int val) {

if (top2 - 1 == top1) {

printf("\nSTACK OVERFLOW");

} else {

top2--;

stack[top2] = val;

printf("%d pushed into the stack", stack[top2]);

}

}

void pop2() {

if (top2 == max) {

printf("\nSTACK UNDERFLOW");

} else {

printf("%d popped from the stack", stack[top2]);

top2++;

}

}

void display2() {

if (top2 == max) {

printf("\nSTACK UNDERFLOW");

} else {

printf("stack:");

for (int i = top2; i < max; i++) {

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

}

}

}

int main() {

int val, choice = 0, op;

while (choice != 3) {

printf("\n\*\* MAIN MENU || MULTIPLE STACKS || ARRAY || C \*\*");

printf("\n1. OPERATIONS FOR STACK1");

printf("\n2. OPERATIONS FOR STACK2");

printf("\n3. EXIT FROM PROGRAM");

printf("\n\nEnter your choice for stack : ");

scanf("%d", &choice);

switch (choice) {

case 1:

op = 0;

while (op != 4) {

printf("\n\n\*\*\* MENU || STACK1 \*\*\*");

printf("\n1. PUSH INTO STACK1");

printf("\n2. POP FROM STACK1");

printf("\n3. DISPLAY STACK1");

printf("\n4. EXIT FROM STACK OPERATIONS");

printf("\n\nEnter your choice of operation : ");

scanf("%d", &op);

switch (op) {

case 1:

printf("\nEnter Value : ");

scanf("%d", &val);

push1(val);

break;

case 2:

pop1();

break;

case 3:

display1();

break;

case 4:

printf("EXIT FROM THE PROGAMS !!\nBACK TO MAIN MENU !!\n");

break;

default:

printf("\nINVALID OPERATION !!\nTRY AGAIN !!\n");

break;

}

}

break;

case 2:

op = 0;

while (op != 4) {

printf("\n\*\*\* MENU || STACK2 \*\*\*");

printf("\n1. PUSH INTO STACK2");

printf("\n2. POP FROM STACK2");

printf("\n3. DISPLAY STACK2");

printf("\n4. EXIT FROM STACK OPERATIONS");

printf("\n\nEnter your choice of operation : ");

scanf("%d", &op);

switch (op) {

case 1:

printf("Enter Value : ");

scanf("%d", &val);

push2(val);

break;

case 2:

pop2();

break;

case 3:

display2();

break;

case 4:

printf("EXIT FROM THE PROGAMS !!\nBACK TO MAIN MENU !!\n");

break;

default:

printf("\nINVALID OPERATION !!\nTRY AGAIN !!\n");

break;

}

}

break;

case 3:

printf("\nEXIT FROM THE PROGRAM !!");

break;

default:

printf("\nINVALID CHOICE !!\nTRY AGAIN !!\n");

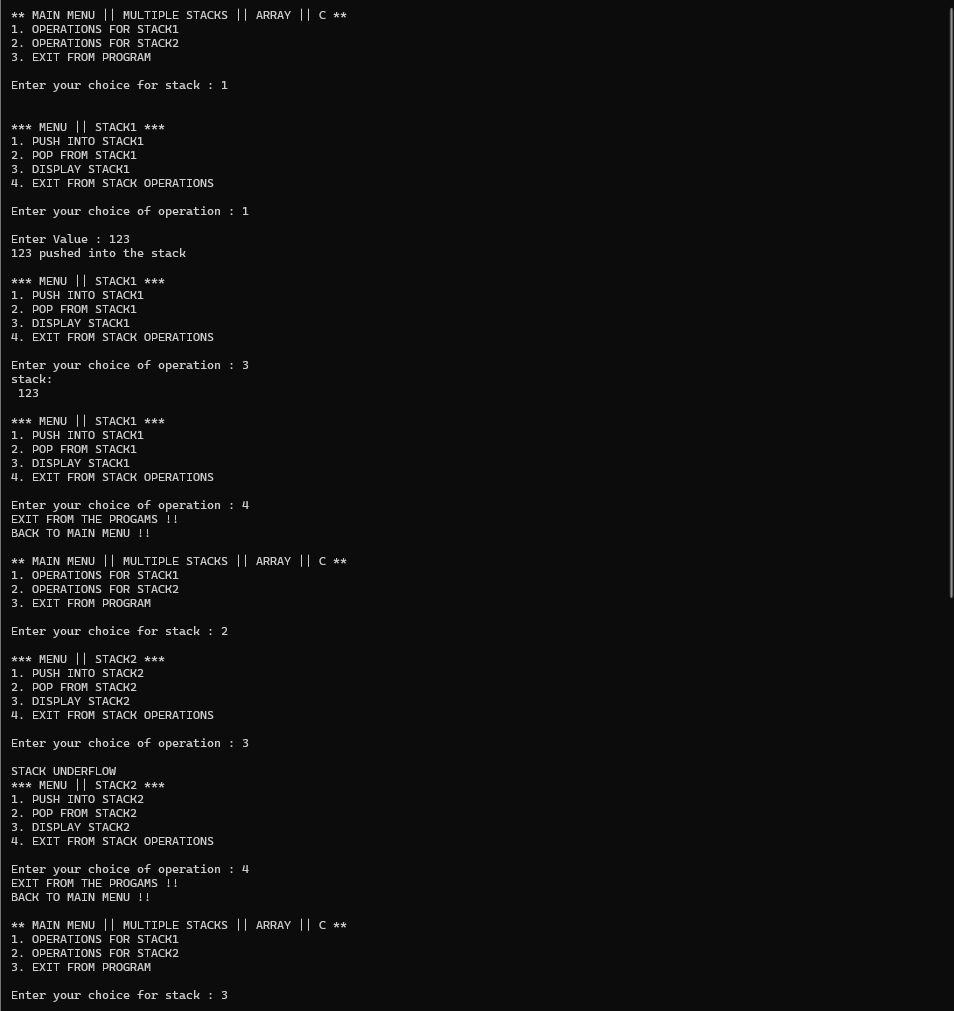
break;

}

}

return 0;

}

****

**9. Repeat the above in C++ using a class-based implementation.**

// 8. Write a program in C to implement two stacks in one array such that:

// o Stack 1 grows from left to right.

// o Stack 2 grows from right to left.

// o The program should support push1(), push2(), pop1(), pop2() and check for

// overflow/underflow conditions.

// 9. Repeat the above in C++ using a class-based implementation.

#include <iostream>

using namespace std;

#define max 10

class mStacks {

public:

int stack[max], top1 = -1, top2 = max;

void push1(int val) {

if (top1 == top2 - 1) {

printf("\nSTACK OVERFLOW");

} else {

top1++;

stack[top1] = val;

printf("%d pushed into the stack", stack[top1]);

}

}

void pop1() {

if (top1 == -1) {

printf("\nSTACK UNDERFLOW");

} else {

printf("%d popped from the stack", stack[top1]);

top1--;

}

}

void display1() {

if (top1 == -1) {

printf("\nSTACK UNDERFLOW");

} else {

printf("stack:");

for (int i = top1; i >= 0; i--) {

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

}

}

}

void push2(int val) {

if (top2 - 1 == top1) {

printf("\nSTACK OVERFLOW");

} else {

top2--;

stack[top2] = val;

printf("%d pushed into the stack", stack[top2]);

}

}

void pop2() {

if (top2 == max) {

printf("\nSTACK UNDERFLOW");

} else {

printf("%d popped from the stack", stack[top2]);

top2++;

}

}

void display2() {

if (top2 == max) {

printf("\nSTACK UNDERFLOW");

} else {

printf("stack:");

for (int i = top2; i < max; i++) {

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

}

}

}

};

int main() {

mStacks stk;

int val, choice = 0, op;

while (choice != 3) {

printf("\n\*\* MAIN MENU || MULTIPLE STACKS || ARRAY || CPP \*\*");

printf("\n1. OPERATIONS FOR STACK1");

printf("\n2. OPERATIONS FOR STACK2");

printf("\n3. EXIT FROM PROGRAM");

printf("\n\nEnter your choice for stack : ");

scanf("%d", &choice);

switch (choice) {

case 1:

op = 0;

while (op != 4) {

printf("\n\n\*\*\* MENU || STACK1 \*\*\*");

printf("\n1. PUSH INTO STACK1");

printf("\n2. POP FROM STACK1");

printf("\n3. DISPLAY STACK1");

printf("\n4. EXIT FROM STACK OPERATIONS");

printf("\n\nEnter your choice of operation : ");

scanf("%d", &op);

switch (op) {

case 1:

printf("\nEnter Value : ");

scanf("%d", &val);

stk.push1(val);

break;

case 2:

stk.pop1();

break;

case 3:

stk.display1();

break;

case 4:

printf("EXIT FROM THE PROGAMS !!\nBACK TO MAIN MENU !!\n");

break;

default:

printf("\nINVALID OPERATION !!\nTRY AGAIN !!\n");

break;

}

}

break;

case 2:

op = 0;

while (op != 4) {

printf("\n\*\*\* MENU || STACK2 \*\*\*");

printf("\n1. PUSH INTO STACK2");

printf("\n2. POP FROM STACK2");

printf("\n3. DISPLAY STACK2");

printf("\n4. EXIT FROM STACK OPERATIONS");

printf("\n\nEnter your choice of operation : ");

scanf("%d", &op);

switch (op) {

case 1:

printf("Enter Value : ");

scanf("%d", &val);

stk.push2(val);

break;

case 2:

stk.pop2();

break;

case 3:

stk.display2();

break;

case 4:

printf("EXIT FROM THE PROGAMS !!\nBACK TO MAIN MENU !!\n");

break;

default:

printf("\nINVALID OPERATION !!\nTRY AGAIN !!\n");

break;

}

}

break;

case 3:

printf("\nEXIT FROM THE PROGRAM !!");

break;

default:

printf("\nINVALID CHOICE !!\nTRY AGAIN !!\n");

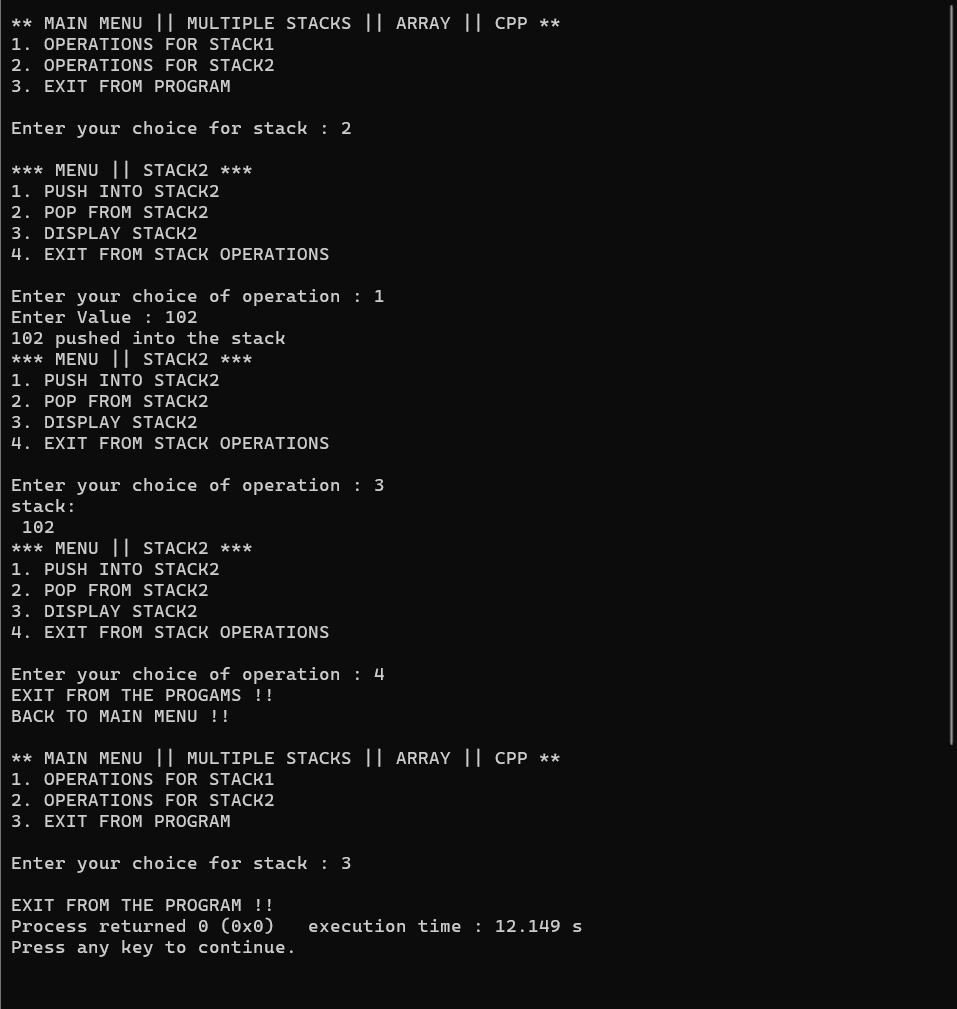
break;

}

}

return 0;

}



## Applications of Stack

10. Write a program to reverse a string using a stack implemented with arrays.

#include <stdio.h>

#include <string.h>

#define MAX 100

char stk[MAX];

int top = -1;

void Push(char c);

char Pop();

int main() {

char str[MAX], rev[MAX];

int i, count = 0;

printf("Enter a String : ");

scanf("%s", str);

printf("\nORIGINAL STRING : %s", str);

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

Push(str[i]);

}

while (top != -1) {

rev[count++] = Pop();

}

rev[count] = '\0';

printf("\n\nREVERSED STRING : %s\n", rev);

return 0;

}

void Push(char c) {

if (top == MAX - 1) {

printf("Stack Overflow\n");

} else {

top++;

stk[top] = c;

}

}

char Pop() {

if (top == -1) {

printf("Stack Underflow\n");

return '\0';

} else {

return stk[top--];

}

}

11. **Write a program to reverse a string in C++ using the stack STL.**

#include <iostream>

#include <string.h>

#include <stack>

using namespace std;

int main()

{

    string str = "", rev = "";

    cout << "Enter a String : ";

    cin >> str;

    stack<char> stk;

    for (char c : str)

    {

        stk.push(c);

    }

    while (!stk.empty())

    {

        rev += stk.top();

        stk.pop();

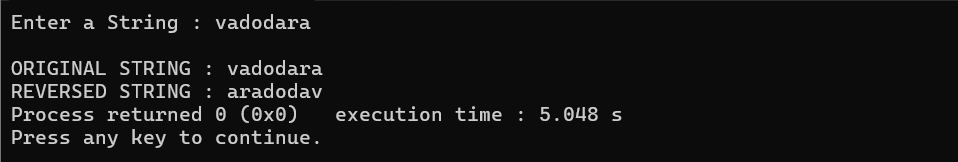
    }

    cout << "\nORIGINAL STRING : " << str;

    cout << "\nREVERSED STRING : " << rev;

    return 0;

}



12. **Write a program in c to reverse an integer array using a stack (implemented with array).**

#include <stdio.h>

#define max 100

int stk[max], top = -1;

void Push(int value);

int Pop();

int main()

{

    int n, i, val, num[max];

    printf("Enter the size of the array : ");

    scanf("%d", &n);

    printf("Enter Values for the array : \n");

    for (i = 0; i < n; i++)

    {

        printf("for index %d : ", i);

        scanf("%d", &num[i]);

    }

    printf("\nORIGINAL ARRAY :");

    for (i = 0; i < n; i++)

    {

        printf("\nValue at index %d = %d", i, num[i]);

    }

    for (i = 0; i < n; i++)

    {

        Push(num[i]);

    }

    for (i = 0; i < n; i++)

    {

        val = Pop();

        num[i] = val;

    }

    printf("\n\nREVERSED ARRAY :");

    for (i = 0; i < n; i++)

    {

        printf("\nValue at index %d = %d", i, num[i]);

    }

    return 0;

}

void Push(int value)

{

    top++;

    stk[top] = value;

}

int Pop()

{

    int val;

    if (top == -1)

    {

        return 0;

    }

    else

    {

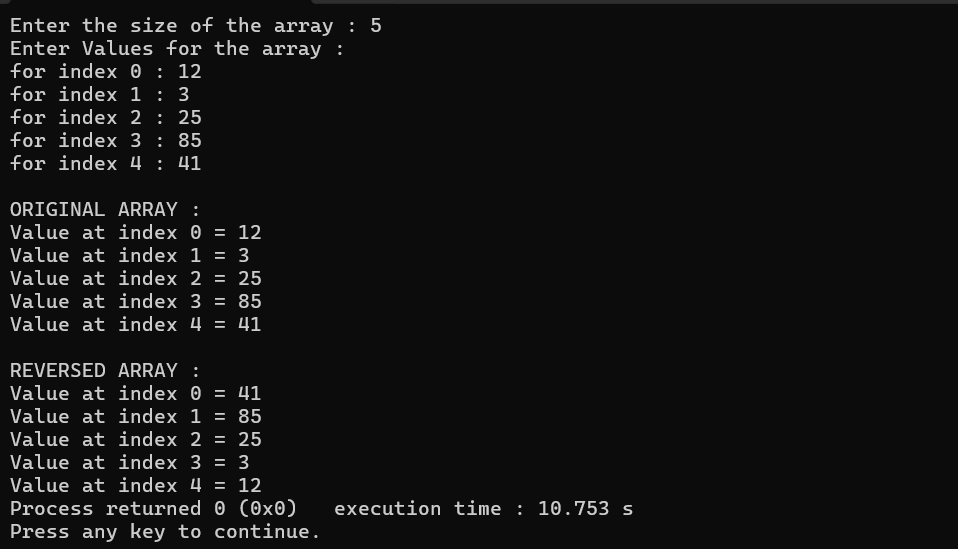
        val = stk[top];

        stk[top--];

        return val;

    }

}



**13. Write a program in C to check whether a given expression has balanced parentheses using a stack (array/linked list).**

#include <stdio.h>

#include <string.h>

#define MAX 20

int top = -1;

char stk[MAX];

void push(char c);

char pop();

int main()

{

    char expr[MAX], tmp;

    int i, flag = 1;

    printf("Enter Expression : ");

    gets(expr);

    for (i = 0; i < strlen(expr); i++)

    {

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

        {

            push(expr[i]);

        }

        else if (expr[i] == ')' || expr[i] == '}' || expr[i] == ']')

        {

            if (top == -1)

            {

                flag = 0;

                break;

            }

            else

            {

                tmp = pop();

                if (expr[i] == ')' && (tmp == '{' || tmp == '['))

                {

                    flag = 0;

                }

                if (expr[i] == '}' && (tmp == '(' || tmp == '['))

                {

                    flag = 0;

                }

                if (expr[i] == ']' && (tmp == '{' || tmp == '('))

                {

                    flag = 0;

                }

            }

        }

    }

    if (top != -1)

    {

        flag = 0;

    }

    if (flag == 1)

    {

        printf("\nValid Expression !!\n");

    }

    else

    {

        printf("\nInvalid Expression !!\n");

    }

    return 0;

}

void push(char c)

{

    if (top == MAX - 1)

    {

        printf("\nSTACK OVERFLOW !!");

    }

    else

    {

        stk[++top] = c;

    }

}

char pop()

{

    if (top == -1)

    {

        printf("\nSTACK UNDERFLOW !!");

        return '\0';

    }

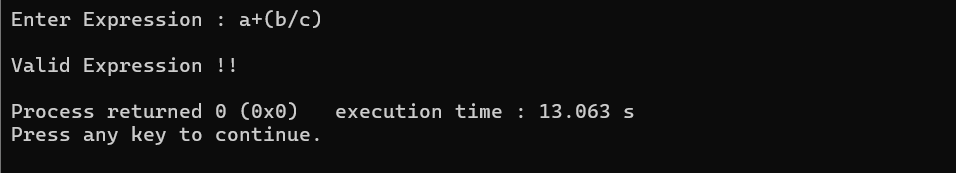
    else

    {

        return stk[top--];

    }

}



14. **Repeat the above using STL stack.in**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

class ParenthesisCheck

{

public:

    stack<char> stk;

    bool isMatchingPair(char open, char close)

    {

        return (open == '(' && close == ')' || open == '{' && close == '}' || open == '[' && close == ']');

    }

    bool isBalanced(const string &expr)

    {

        for (char c : expr)

        {

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

            {

                stk.push(c);

            }

            else if (c == ')' || c == '}' || c == ']')

            {

                if (stk.empty())

                    return false;

                char open = stk.top();

                stk.pop();

                if (!isMatchingPair(open, c))

                    return false;

            }

        }

        return stk.empty();

    }

};

int main()

{

    ParenthesisCheck checker;

    string expr;

    cout << "enter the expression : ";

    cin >> expr;

    if (checker.isBalanced(expr))

    {

        cout << "balanced parenthesis" << endl;

    }

    else

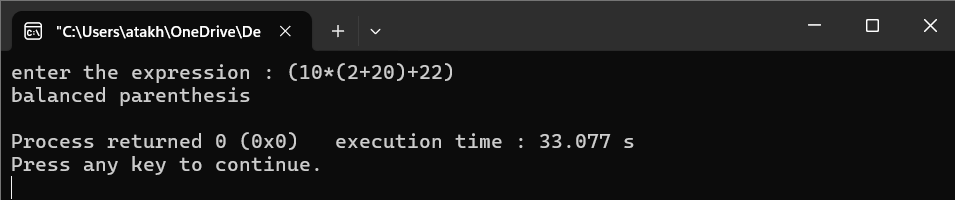
    {

        cout << "unbalanced parenthesis" << endl;

    }

    return 0;

}



**15. Write a program in C to convert an infix expression to a postfix expression using a stack.**

#include <stdio.h>

#include <string.h>

#define max 100

char stk[max];

int top = -1;

void Push(char);

char Pop();

void InfixToPostfix(char source[], char target[]);

int priority(char);

int main()

{

    char infix[max], postfix[max];

    printf("Enter an Expression : ");

    gets(infix);

    strcpy(postfix, "");

    InfixToPostfix(infix, postfix);

    printf("\nPostfix of given Expression : ");

    puts(postfix);

    return 0;

}

void InfixToPostfix(char source[], char target[])

{

    int i = 0, j = 0;

    char tmp;

    strcpy(target, "");

    while (source[i] != '\0')

    {

        if (source[i] == '(')

        {

            Push(source[i]);

            i++;

        }

        else if (source[i] == ')')

        {

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

            {

                target[j] = Pop();

                j++;

            }

            if (top == -1)

            {

                printf("\nInvalid Expression !!");

                exit(1);

            }

            tmp = Pop();

            i++;

        }

        else if (isdigit(source[i]) || isalpha(source[i]))

        {

            target[j] = source[i];

            j++;

            i++;

        }

        else if (source[i] == '+' || source[i] == '-' || source[i] == '\*' || source[i] == '/' || source[i] == '%')

        {

            while ((top != -1) && (stk[top] != '(') && (priority(stk[top]) > priority(source[i])))

            {

                target[j] = Pop();

                j++;

            }

            Push(source[i]);

            i++;

        }

        else

        {

            printf("\n Invalid Element in Expression !!");

            exit(1);

        }

    }

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

    {

        target[j] = Pop();

        j++;

    }

    target[j] = '\0';

}

int priority(char op)

{

    if (op == '/' || op == '\*' || op == '%')

    {

        return 1;

    }

    else if (op == '+' || op == '-')

    {

        return 0;

    }

}

void Push(char c)

{

    if (top == max)

    {

        printf("\nSTACK OVERFLOW !!");

    }

    else

    {

        top++;

        stk[top] = c;

    }

}

char Pop()

{

    char val = ' ';

    if (top == -1)

    {

        printf("\nSTACK UNDERFLOW !!");

    }

    else

    {

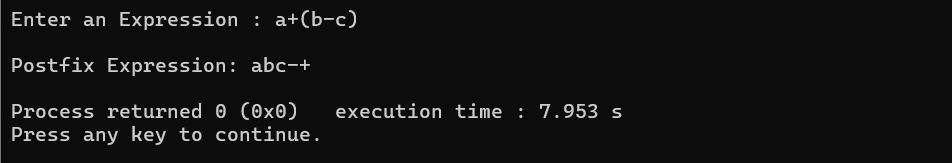
        val = stk[top];

        top--;

    }

    return val;

}



16. **Repeat the above in C++ using STL stack.**

#include <iostream>

#include <cstring>

#include <stack>

#include <cctype>

using namespace std;

int top = -1;

int priority(char op)

{

    if (op == '/' || op == '+' || op == '%')

    {

        return 1;

    }

    else if (op == '+' || op == '-')

    {

        return 0;

    }

}

void InfixToPostfix(const char source[], char target[])

{

    stack<char> stk;

    int i = 0, j = 0;

    while (source[i] != '\0')

    {

        char ch = source[i];

        if (isalnum(ch))

        {

            target[j++] = ch;

        }

        else if (ch == '(')

        {

            stk.push(ch);

        }

        else if (ch == ')')

        {

            while (!stk.empty() && stk.top() != '(')

            {

                target[j++] = stk.top();

                stk.pop();

            }

            if (stk.empty())

            {

                cout << "\nInvalid Expression !!";

                exit(1);

            }

            stk.pop();

        }

        else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '%')

        {

            while (!stk.empty() && stk.top() != '(' && priority(stk.top()) >= priority(ch))

            {

                target[j++] = stk.top();

                stk.pop();

            }

            stk.push(ch);

        }

        else

        {

            cout << "\nInvalid Element in Expression !!";

            exit(1);

        }

        i++;

    }

    while (!stk.empty())

    {

        if (stk.top() == '(')

        {

            cout << "\nInvalid Expression !!";

            exit(1);

        }

        target[j++] = stk.top();

        stk.pop();

    }

    target[j] = '\0';

}

int main()

{

    char infix[100], postfix[100];

    cout << "Enter an Expression : ";

    cin >> infix;

    strcpy(postfix, "");

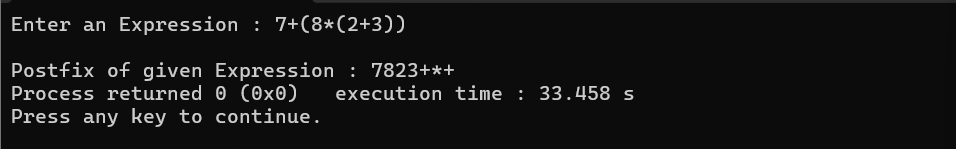
    InfixToPostfix(infix, postfix);

    cout << "\nPostfix of given Expression : ";

    cout << postfix;

    return 0;

}



**17. Write a program in C to convert an infix expression to a prefix expression using a stack.**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#define max 100

char stk[max];

int top = -1;

void ReverseExpr(char str[]);

void push(char);

char pop();

void InfixToPostfix(char source[], char target[]);

int Priority(char);

char infix[max], postfix[max], tmp[max];

int main()

{

    printf("Enter an Expression : ");

    fgets(infix, max, stdin);

    infix[strcspn(infix, "\n")] = '\0'; // remove newline

    ReverseExpr(infix);

    strcpy(postfix, "");

    InfixToPostfix(tmp, postfix);

    printf("\nPostfix Representation = ");

    puts(postfix);

    strcpy(tmp, "");

    ReverseExpr(postfix);

    printf("\nPrefix Representation = ");

    puts(tmp);

    return 0;

}

void ReverseExpr(char str[])

{

    int len = strlen(str), i = 0, j = len - 1;

    while (j >= 0)

    {

        if (str[j] == '(')

            tmp[i] = ')';

        else if (str[j] == ')')

            tmp[i] = '(';

        else

            tmp[i] = str[j];

        i++;

        j--;

    }

    tmp[i] = '\0';

}

void InfixToPostfix(char source[], char target[])

{

    int i = 0, j = 0;

    char temp;

    strcpy(target, "");

    while (source[i] != '\0')

    {

        if (source[i] == '(')

        {

            push(source[i]);

            i++;

        }

        else if (source[i] == ')')

        {

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

            {

                target[j++] = pop();

            }

            if (top == -1)

            {

                printf("\nIncorrect Expression !!");

                exit(1);

            }

            temp = pop();

            i++;

        }

        else if (isdigit(source[i]) || isalpha(source[i]))

        {

            target[j++] = source[i++];

        }

        else if (source[i] == '+' || source[i] == '-' || source[i] == '\*' || source[i] == '/' || source[i] == '%')

        {

            while ((top != -1) && (stk[top] != '(') && (Priority(stk[top]) >= Priority(source[i])))

            {

                target[j++] = pop();

            }

            push(source[i]);

            i++;

        }

        else

        {

            printf("\nIncorrect Element in the Expression !!");

            exit(1);

        }

    }

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

    {

        target[j++] = pop();

    }

    target[j] = '\0';

}

int Priority(char op)

{

    if (op == '/' || op == '\*' || op == '%')

        return 1;

    else if (op == '+' || op == '-')

        return 0;

    return -1;

}

void push(char val)

{

    if (top == max - 1)

        printf("\n STACK OVERFLOW");

    else

        stk[++top] = val;

}

char pop()

{

    if (top == -1)

    {

        printf("\n STACK UNDERFLOW");

        return ' ';

    }

    return stk[top--];

`}

18. Repeat the above in C++ using STL stack.

#include <iostream>

#include <stack>

#include <cstring>

#include <cctype>

using namespace std;

#define MAX 100

void ReverseExpr(const char str[], char result[]);

void InfixToPostfix(const char source[], char target[]);

int Priority(char);

char infix[MAX], postfix[MAX], tmp[MAX];

int main()

{

    cout << "Enter an Expression : ";

    cin.getline(infix, MAX);

    ReverseExpr(infix, tmp);

    strcpy(postfix, "");

    InfixToPostfix(tmp, postfix);

    cout << "\nPostfix Representation = " << postfix << endl;

    strcpy(tmp, "");

    ReverseExpr(postfix, tmp);

    cout << "\nPrefix Representation = " << tmp << endl;

    return 0;

}

void ReverseExpr(const char str[], char result[])

{

    int len = strlen(str), i = 0, j = len - 1;

    while (j >= 0)

    {

        if (str[j] == '(')

            result[i] = ')';

        else if (str[j] == ')')

            result[i] = '(';

        else

            result[i] = str[j];

        i++;

        j--;

    }

    result[i] = '\0';

}

void InfixToPostfix(const char source[], char target[])

{

    stack<char> stk;

    int i = 0, j = 0;

    while (source[i] != '\0')

    {

        if (source[i] == '(')

        {

            stk.push(source[i]);

            i++;

        }

        else if (source[i] == ')')

        {

            while (!stk.empty() && stk.top() != '(')

            {

                target[j++] = stk.top();

                stk.pop();

            }

            if (stk.empty())

            {

                cout << "\nIncorrect Expression !!";

                exit(1);

            }

            stk.pop();

            i++;

        }

        else if (isdigit(source[i]) || isalpha(source[i]))

        {

            target[j++] = source[i++];

        }

        else if (source[i] == '+' || source[i] == '-' || source[i] == '\*' ||

                 source[i] == '/' || source[i] == '%')

        {

            while (!stk.empty() && stk.top() != '(' &&

                   Priority(stk.top()) >= Priority(source[i]))

            {

                target[j++] = stk.top();

                stk.pop();

            }

            stk.push(source[i]);

            i++;

        }

        else

        {

            cout << "\nIncorrect Element in the Expression !!";

            exit(1);

        }

    }

    while (!stk.empty() && stk.top() != '(')

    {

        target[j++] = stk.top();

        stk.pop();

    }

    target[j] = '\0';

}

int Priority(char op)

{

    if (op == '/' || op == '\*' || op == '%')

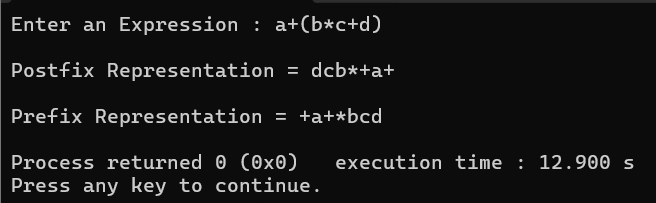
        return 1;

    else if (op == '+' || op == '-')

        return 0;

    return -1;

}



19. **Write a program in C to evaluate a postfix expression.**

#include <stdio.h>

#include <ctype.h>

#define max 100

float stk[max];

int top = -1;

void Push(float val);

float Pop();

float EvaluatePostfix(char expr[]);

int main()

{

    float value;

    char expr[max];

    printf("Enter Postfix Expression : ");

    gets(expr);

    value = EvaluatePostfix(expr);

    printf("\nEvaluation of the Postfix Expression = %.2f", value);

    return 0;

}

float EvaluatePostfix(char expr[])

{

    int i = 0;

    float op1, op2, value;

    while (expr[i] != '\0')

    {

        if (isdigit(expr[i]))

        {

            Push((float)(expr[i] - '0'));

        }

        else

        {

            op2 = Pop();

            op1 = Pop();

            switch (expr[i])

            {

            case '+':

                value = op1 + op2;

                break;

            case '-':

                value = op1 - op2;

                break;

            case '/':

                value = op1 / op2;

                break;

            case '\*':

                value = op1 \* op2;

                break;

            case '%':

                value = (int)op1 % (int)op2;

                break;

            default:

                break;

            }

            Push(value);

        }

        i++;

    }

    return (Pop());

}

void Push(float value)

{

    if (top == max - 1)

    {

        printf("\nSTACK OVERFLOW");

    }

    else

    {

        top++;

        stk[top] = value;

    }

}

float Pop()

{

    float value = -1;

    if (top == -1)

    {

        printf("\nSTACK UNDERFLOW");

    }

    else

    {

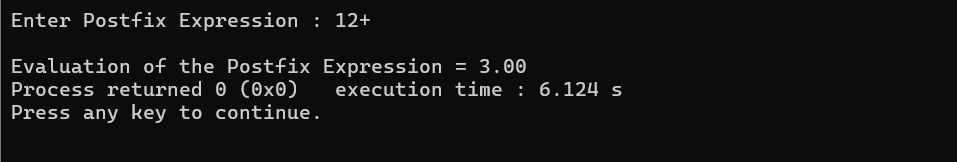
        value = stk[top];

        top--;

    }

    return value;

}



20. **Repeat the above in C++ using STL stack.**

#include <iostream>

#include <stack>

#include <cstring>

#include <cctype>

using namespace std;

#define max 100

float EvaluatePostfix(char expr[]);

int main()

{

    float value;

    char expr[max];

    cout << "Enter Postfix Expression : ";

    cin >> expr;

    value = EvaluatePostfix(expr);

    cout << "\nEvaluation of the Postfix Expression = " << value;

    return 0;

}

float EvaluatePostfix(char expr[])

{

    stack<float> stk;

    int i = 0;

    float op1, op2, value;

    while (expr[i] != '\0')

    {

        if (isdigit(expr[i]))

        {

            stk.push((float)(expr[i] - '0'));

        }

        else

        {

            op2 = stk.top();

            stk.pop();

            op1 = stk.top();

            stk.pop();

            switch (expr[i])

            {

            case '+':

                value = op1 + op2;

                break;

            case '-':

                value = op1 - op2;

                break;

            case '/':

                value = op1 / op2;

                break;

            case '\*':

                value = op1 \* op2;

                break;

            case '%':

                value = (int)op1 % (int)op2;

                break;

            default:

                break;

            }

            stk.push(value);

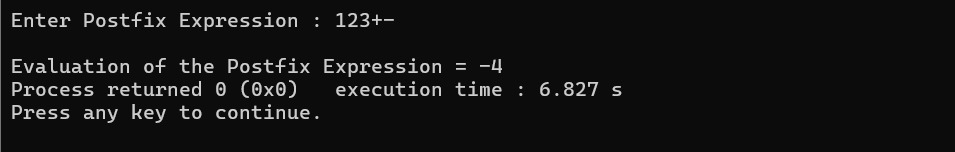
        }

        i++;

    }

    return (stk.top());

}



21. **Write a program in C to evaluate a prefix expression.**

#include <stdio.h>

#include <string.h>

#define max 100

int stk[max];

int top = -1;

void Push(int);

int Pop();

int get\_type(char);

int main()

{

    char expr[max];

    int len, value, i, op1, op2, res;

    printf("Enter Prefix Expression : ");

    gets(expr);

    len = strlen(expr);

    for (i = len - 1; i >= 0; i--)

    {

        switch (get\_type(expr[i]))

        {

        case 0:

            value = expr[i] - '0';

            Push(value);

            break;

        case 1:

            op1 = Pop();

            op2 = Pop();

            switch (expr[i])

            {

            case '+':

                res = op1 + op2;

                break;

            case '-':

                res = op1 - op2;

                break;

            case '/':

                res = op1 / op2;

                break;

            case '\*':

                res = op1 \* op2;

                break;

            case '%':

                res = op1 % op2;

                break;

            }

            Push(res);

        }

    }

    printf("\nResult = %d", Pop());

    return 0;

}

void Push(int value)

{

    top++;

    stk[top] = value;

}

int Pop()

{

    return (stk[top--]);

}

int get\_type(char c)

{

    if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '%')

    {

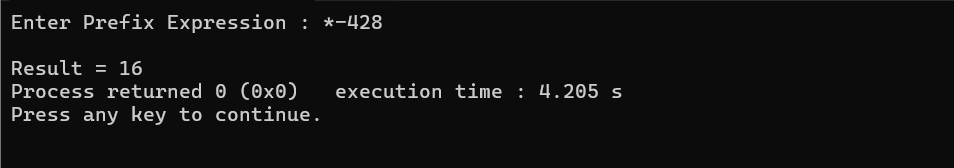
        return 1;

    }

    else

        return 0;

}



22. **Repeat the above in C++ using STL stack.**

#include <iostream>

#include <stack>

#include <cstring>

#include <cctype>

using namespace std;

#define max 100

int get\_type(char);

int main()

{

    char expr[max];

    stack<int> stk;

    int len, value, i, op1, op2, res;

    cout << "Enter Prefix Expression : ";

    cin >> expr;

    len = strlen(expr);

    for (i = len - 1; i >= 0; i--)

    {

        switch (get\_type(expr[i]))

        {

        case 0:

            value = expr[i] - '0';

            stk.push(value);

            break;

        case 1:

            op1 = stk.top();

            stk.pop();

            op2 = stk.top();

            stk.pop();

            switch (expr[i])

            {

            case '+':

                res = op1 + op2;

                break;

            case '-':

                res = op1 - op2;

                break;

            case '/':

                res = op1 / op2;

                break;

            case '\*':

                res = op1 \* op2;

                break;

            case '%':

                res = op1 % op2;

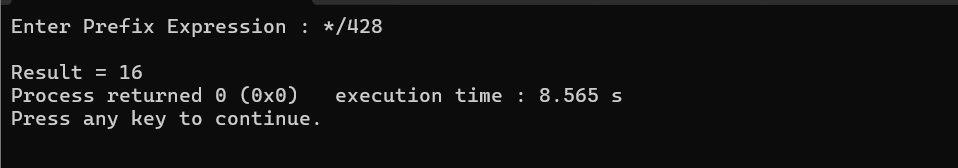
                break;

            }

            stk.push(res);

        }

    }

    cout << "\nResult = " << stk.top();

    return 0;

}

int get\_type(char c)

{

    if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '%')

    {

        return 1;

    }

    else

        return 0;

}

s