ASSIGNMENT 3: STACK

**Q1.**

#include <iostream>

using namespace std;

//menu driven stack program

int a[5];

int top=-1;

int n=sizeof(a)/sizeof(a[0]);

void push(int value){

if (top==n-1){

cout<<"Stack Overflow";

}

else{

cout<<"Enter elements: ";

cin>>value;

a[++top]=value;

cout << value << " pushed";

}

}

void pop(){

if (top==-1){

cout<<"Stack Underflow";

}

else{

cout<<a[top];

top--;

}

}

bool isEmpty(){

return (top==-1);

}

bool isFull(){

return (top==n-1);

}

void display(){

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

cout<<a[i]<<" ";

}

}

void peek(){

if (isEmpty()){

cout<<"Stack is empty";

}

else{

cout<<"Top element is "<<a[top];

}

}

int main(){

int choice;

int value;

do{

cout<<"\nMENU\n";

cout<<"1.PUSH\n2.POP\n3.ISEMPTY\n4.ISFULL\n5.DISPLAY\n6.PEEK\n7.EXIT\n";

cout<<"Enter your choice: ";

cin>>choice;

switch (choice){

case 1:

push(value);

break;

case 2:

pop();

break;

case 3:

if (isEmpty()){

cout<<"Stack is empty";

}

else{

cout<<"Stack is not empty";

}

break;

case 4:

if (isFull()){

cout<<"Stack is full";

}

else{

cout<<"Stack is not full";

}

break;

case 5:

display();

break;

case 6:

peek();

break;

case 7:

cout<<"Exiting...";

break;

default:

cout<<"INVALID!!!!";

break;

}

} while (choice != 7);

return 0;

}

**Q2.**

#include <iostream>

using namespace std;

//reverse a string using stack

char str[]="DataStructure";

int top=-1;

void pop(){

if (top==-1){

cout<<"Underflow";

}

else{

cout<<str[top];

top--;

}

}

int main(){

top=sizeof(str)/sizeof(str[0])-1;

cout<<"Reversed string is ";

while(top!=-1){

pop();

}

cout<<endl;

return 0;

}

**Q3.**

#include <iostream>

using namespace std;

int a[5];

int top=-1;

int n=sizeof(a)/sizeof(a[0]);

void push(char c){

if (top==n-1){

cout<<"Stack Overflow\n";

}

else{

a[++top]=c;

}

}

char pop(){

if (top==-1){

return '\0';

}

else{

return a[top--];

}

}

bool isEmpty(){

return (top==-1);

}

bool isBalanced(string expr){

for (int i=0;i<expr.length();i++){

char ch=expr[i];

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

push(ch);

}

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

if(isEmpty()){

return false;

}

char topChar=pop();

if ((ch == ')' && topChar != '(') ||

(ch == '}' && topChar != '{') ||

(ch == ']' && topChar != '[')) {

return false;

}

}

}

return isEmpty();

}

int main(){

string expr;

cout << "Enter an expression: ";

getline(cin, expr);

if (isBalanced(expr))

cout << "\nBalanced\n";

else

cout << "\nNot Balanced\n";

return 0;

}

**Q3.**

#include <iostream>

using namespace std;

int a[5];

int top=-1;

int n=sizeof(a)/sizeof(a[0]);

void push(char c){

if (top==n-1){

cout<<"Stack Overflow\n";

}

else{

a[++top]=c;

}

}

char pop(){

if (top==-1){

return '\0';

}

else{

return a[top--];

}

}

bool isEmpty(){

return (top==-1);

}

bool isBalanced(string expr){

for (int i=0;i<expr.length();i++){

char ch=expr[i];

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

push(ch);

}

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

if(isEmpty()){

return false;

}

char topChar=pop();

if ((ch == ')' && topChar != '(') ||

(ch == '}' && topChar != '{') ||

(ch == ']' && topChar != '[')) {

return false;

}

}

}

return isEmpty();

}

int main(){

string expr;

cout << "Enter an expression: ";

getline(cin, expr);

if (isBalanced(expr))

cout << "\nBalanced\n";

else

cout << "\nNot Balanced\n";

return 0;

}

**Q4.**

#include <iostream>

#include <stack>

#include <cctype>

using namespace std;

// Function to return precedence of operators

int precedence(char op) {

if (op == '^') return 3;

if (op == '\*' || op == '/') return 2;

if (op == '+' || op == '-') return 1;

return -1;

}

// Function to check if operator is right-associative

bool isRightAssociative(char op) {

return op == '^';

}

// Function to convert Infix to Postfix

string infixToPostfix(string infix) {

stack<char> st;

string postfix = "";

for (char c : infix) {

// If operand, add to output

if (isalnum(c)) {

postfix += c;

}

// If '(', push to stack

else if (c == '(') {

st.push(c);

}

// If ')', pop until '('

else if (c == ')') {

while (!st.empty() && st.top() != '(') {

postfix += st.top();

st.pop();

}

if (!st.empty()) st.pop(); // remove '('

}

// If operator

else {

while (!st.empty() && precedence(c) <= precedence(st.top()) &&

!(isRightAssociative(c) && precedence(c) == precedence(st.top()))) {

postfix += st.top();

st.pop();

}

st.push(c);

}

}

// Pop all remaining operators

while (!st.empty()) {

postfix += st.top();

st.pop();

}

return postfix;

}

int main() {

string infix;

cout << "Enter infix expression: ";

cin >> infix;

string postfix = infixToPostfix(infix);

cout << "Postfix Expression: " << postfix << endl;

return 0;

}

**Q5.**

#include <iostream>

#include <stack>

#include <cctype>

using namespace std;

// Function to evaluate postfix expression

int evaluatePostfix(string postfix) {

stack<int> st;

for (char c : postfix) {

// If operand, push to stack

if (isdigit(c)) {

st.push(c - '0'); // convert char → int

}

else {

// Operator: pop two operands

int val2 = st.top(); st.pop();

int val1 = st.top(); st.pop();

switch (c) {

case '+': st.push(val1 + val2); break;

case '-': st.push(val1 - val2); break;

case '\*': st.push(val1 \* val2); break;

case '/': st.push(val1 / val2); break;

case '^': {

int result = 1;

for (int i = 0; i < val2; i++) result \*= val1;

st.push(result);

break;

}

}

}

}

return st.top();

}

int main() {

string postfix;

cout << "Enter postfix expression (single-digit operands): ";

cin >> postfix;

cout << "Evaluation Result: " << evaluatePostfix(postfix) << endl;

return 0;

}