*1. . Develop a menu driven program demonstrating the following operations on a Stack using array: (i) push(), (ii) pop(), (iii) isEmpty(), (iv) isFull(), (v) display(), and (vi) peek().*

#include<iostream>

using namespace std;

class stack{

public:

int top = -1;

int arr[10];

int n = 3;

stack(){

cout<<"Stack is created: ";

}

void push(int k){

if(top == n-1){

cout<<"\nStack Overflow!";

}

else{

top = top +1;

arr[top] = k;

}

}

void pop(){

if(top == -1){

cout<<"Underflow!";

}

else{

top = top -1;

}

}

int peek(){

return arr[top];

}

void isEmpty(){

if(top == -1){

cout<<"\nStack is empty. ";

}

else {

cout<<"\nStack is not empty. ";

}

}

void isFull(){

if(top == n-1){

cout<<"\nStack is full. ";

}

else{

cout<<"\nStack is not full. ";

}

}

int topE(){

return arr[top];

}

};

int main(){

stack s;

s.push(0);

s.push(1);

s.push(2);

int dis[10];

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

int num = s.topE();

dis[i] = num;

cout<<"\n"<<dis[i];

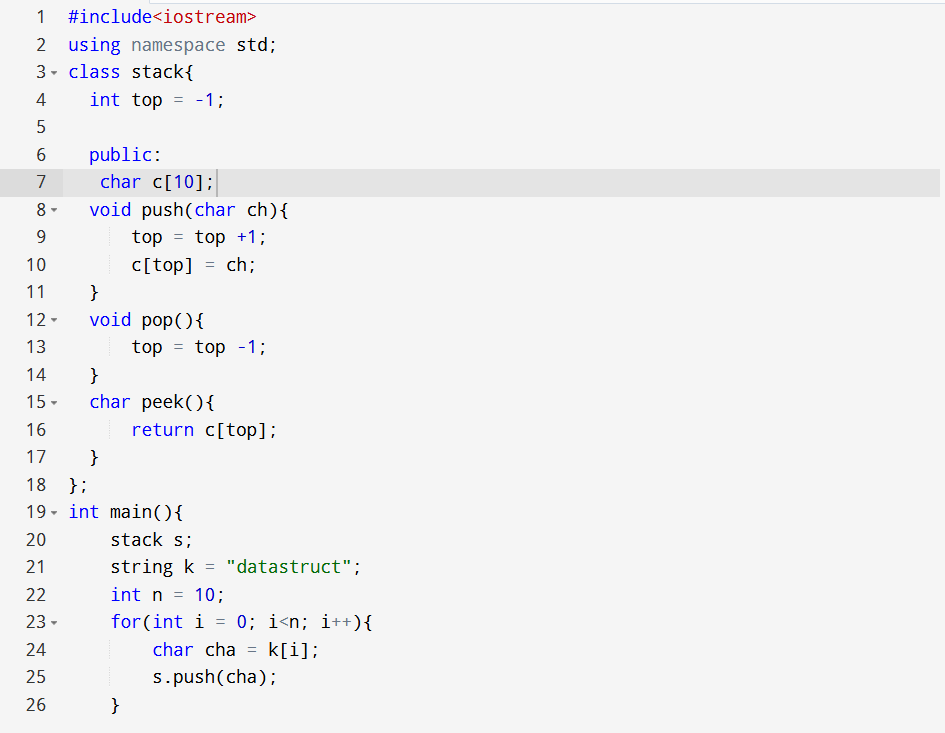
s.top--;

}

s.top = s.n;

}

*2. Given a string, reverse it using STACK. For example “DataStructure” should be output as “erutcurtSataD.”*



*3. Write a program that checks if an expression has balanced parentheses.*

#include <iostream>

#include <stack>

#include <vector>

#include <string>

using namespace std;

bool isBalanced(string& s) {

stack<char> st;

for (char c : s) {

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

st.push(c);

}

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

// No opening bracket

if (st.empty()) return false;

char top = st.top();

if ((c == ')' && top != '(') ||

(c == '}' && top != '{') ||

(c == ']' && top != '[')) {

return false;

}

st.pop();

}

}

return st.empty();

}

int main() {

string s="[()()]{}";

cout<<(isBalanced(s)?"true":"false");

return 0;

}

*4. Write a program to convert an Infix expression into a Postfix expression*

#include <iostream>

#include <stack>

#include <string>

using namespace std;

int precedence(char op) {

if (op == '+' || op == '-')

return 1;

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

return 2;

if (op == '^')

return 3;

return 0;

}

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

string infixToPostfix(string infix) {

stack<char> st;

string postfix = "";

for (char c : infix) {

// If operand, add it to output

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {

postfix += c;

}

else if (c == '(') {

st.push(c);

}

else if (c == ')') {

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

postfix += st.top();

st.pop();

}

if (!st.empty())

st.pop(); // Remove '(' from stack

}

else if (isOperator(c)) {

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

postfix += st.top();

st.pop();

}

st.push(c);

}

}

while (!st.empty()) {

postfix += st.top();

st.pop();

}

return postfix;

}

int main() {

string infixExp;

cout << "Enter an infix expression: ";

cin >> infixExp;

string postfixExp = infixToPostfix(infixExp);

cout << "Postfix expression: " << postfixExp << endl;

return 0;

}

*5. Write a program for the evaluation of a Postfix expression.*

#include <iostream>

#include <stack>

#include <string>

#include <cctype>

using namespace std;

int applyOperation(int a, int b, char op) {

switch(op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/':

if(b == 0) {

cout << "Error: Division by zero\n";

exit(EXIT\_FAILURE);

}

return a / b;

default: return 0;

}

}

int evaluatePostfix(string postfix) {

stack<int> st;

for (char c : postfix) {

// If operand, push it to stack (assuming single digit operands)

if (isdigit(c)) {

st.push(c - '0');

}

else {

if (st.size() < 2) {

cout << "Error: Invalid postfix expression\n";

exit(EXIT\_FAILURE);

}

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

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

int result = applyOperation(val1, val2, c);

st.push(result);

}

}

if (st.size() != 1) {

cout << "Error: Invalid postfix expression\n";

exit(EXIT\_FAILURE);

}

return st.top();

}

int main() {

string postfixExp;

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

cin >> postfixExp;

int result = evaluatePostfix(postfixExp);

cout << "Result: " << result << endl;

return 0;

}