**VISHAW BANSAL**

**1024030356**

***ASSIGNMENT-3***

Ans 1

#include <iostream>

using namespace std;

#define MAX 100

class Stack {

int arr[MAX];

int top;

int size;

public:

Stack(int s) {

top = -1;

size = s;

}

bool isEmpty() {

return top == -1;

}

bool isFull() {

return top == size - 1;

}

void push(int x) {

if (isFull()) {

cout << "Stack is full\n";

} else {

arr[++top] = x;

cout << x << " pushed to stack\n";

}

}

void pop() {

if (isEmpty()) {

cout << "Stack is empty\n";

} else {

cout << arr[top--] << " popped from stack\n";

}

}

void peek() {

if (isEmpty()) {

cout << "Stack is empty\n";

} else {

cout << "Top element: " << arr[top] << endl;

}

}

void display() {

if (isEmpty()) {

cout << "Stack is empty\n";

} else {

cout << "Stack elements:\n";

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

cout << arr[i] << " ";

}

cout << endl;

}

}

};

int main() {

int size;

cout << "Enter stack size (max 100): ";

cin >> size;

Stack s(size);

int choice, value;

while (true) {

cout << "\n1. Push\n2. Pop\n3. Peek\n4. isEmpty\n5. isFull\n6. Display\n7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to push: ";

cin >> value;

s.push(value);

break;

case 2:

s.pop();

break;

case 3:

s.peek();

break;

case 4:

cout << (s.isEmpty() ? "Stack is empty\n" : "Stack is not empty\n");

break;

case 5:

cout << (s.isFull() ? "Stack is full\n" : "Stack is not full\n");

break;

case 6:

s.display();

break;

case 7:

return 0;

default:

cout << "Invalid choice\n";

}

}

return 0;

}

****

**Ans 2**

#include <iostream>

#include <stack>

using namespace std;

void reverseUsingStack(string str) {

stack<char> ch;

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

ch.push(str[i]);

}

while (!ch.empty()) {

cout << ch.top();

ch.pop();

}

cout << endl;

}

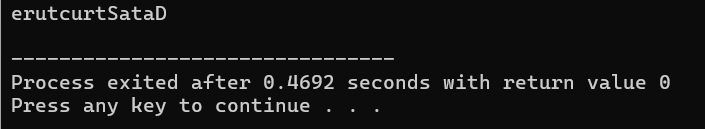
int main() {

string x = "DataStructure";

reverseUsingStack(x);

return 0;

}



**Ans 3**

#include <iostream>

#include <stack>

using namespace std;

bool isMatchingPair(char open, char close) {

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

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

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

}

bool isBalanced(string expr) {

stack<char> s;

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

char ch = expr[i];

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

s.push(ch);

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

if (s.empty() || !isMatchingPair(s.top(), ch)) {

return false;

}

s.pop();

}

}

return s.empty();

}

int main() {

string expression;

cout << "Enter an expression: ";

cin >> expression;

if (isBalanced(expression)) {

cout << "Balanced\n";

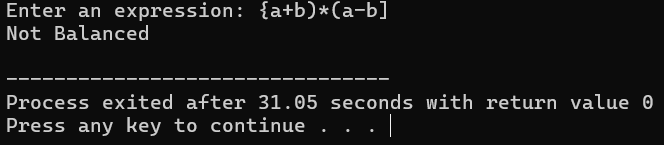
} else {

cout << "Not Balanced\n";

}

return 0;

}



**Ans 4**

#include <iostream>

#include <stack>

using namespace std;

int precedence(char op) {

if (op == '^')

return 3;

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

return 2;

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

return 1;

else

return 0;

}

bool isRightAssociative(char op) {

return op == '^';

}

bool isOperator(char ch) {

return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^';

}

string infixToPostfix(string infix) {

stack<char> s;

string postfix = "";

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

char ch = infix[i];

if (isspace(ch)) continue;

if (isalnum(ch)) {

postfix += ch;

}

else if (ch == '(') {

s.push(ch);

}

else if (ch == ')') {

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

postfix += s.top();

s.pop();

}

if (!s.empty()) s.pop(); // Pop '('

}

else if (isOperator(ch)) {

while (!s.empty() && (

(precedence(ch) < precedence(s.top())) ||

(precedence(ch) == precedence(s.top()) && !isRightAssociative(ch))

) && s.top() != '(') {

postfix += s.top();

s.pop();

}

s.push(ch);

}

}

while (!s.empty()) {

postfix += s.top();

s.pop();

}

return postfix;

}

int main() {

string infix;

cout << "Enter infix expression: ";

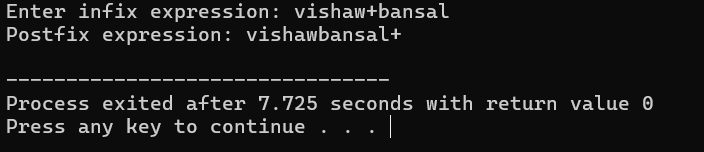
cin >> infix;

string postfix = infixToPostfix(infix);

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

return 0;

}



**Ans 5**

#include <iostream>

#include <stack>

#include <cctype>

using namespace std;

int evaluatePostfix(string expr) {

stack<int> s;

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

char ch = expr[i];

if (isspace(ch)) continue;

if (isdigit(ch)) {

s.push(ch - '0'); // convert char to int

}

else {

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

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

int result;

switch (ch) {

case '+': result = val2 + val1; break;

case '-': result = val2 - val1; break;

case '\*': result = val2 \* val1; break;

case '/': result = val2 / val1; break;

default:

cout << "Invalid operator: " << ch << endl;

return -1;

}

s.push(result);

}

}

return s.top();

}

int main() {

string postfix;

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

cin >> postfix;

int result = evaluatePostfix(postfix);

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

return 0;

}

