|  |  |
| --- | --- |
| Assignment 3 solutions | Ishaan Mishra  1024030346 |

# Stack Programs in C++

## 1. Menu driven program demonstrating stack operations using array

#include <iostream>  
using namespace std;  
  
#define MAX 100  
  
class Stack {  
 int arr[MAX];  
 int top;  
  
public:  
 Stack() { top = -1; }  
  
 void push(int x) {  
 if (top == MAX - 1) {  
 cout << "Stack Overflow\n";  
 return;  
 }  
 arr[++top] = x;  
 cout << x << " pushed into stack\n";  
 }  
  
 void pop() {  
 if (top == -1) {  
 cout << "Stack Underflow\n";  
 return;  
 }  
 cout << arr[top--] << " popped from stack\n";  
 }  
  
 bool isEmpty() {  
 return top == -1;  
 }  
  
 bool isFull() {  
 return top == MAX - 1;  
 }  
  
 void display() {  
 if (top == -1) {  
 cout << "Stack is empty\n";  
 return;  
 }  
 cout << "Stack elements: ";  
 for (int i = top; i >= 0; i--)  
 cout << arr[i] << " ";  
 cout << endl;  
 }  
  
 void peek() {  
 if (top == -1) {  
 cout << "Stack is empty\n";  
 return;  
 }  
 cout << "Top element is: " << arr[top] << endl;  
 }  
};  
  
int main() {  
 Stack s;  
 int choice, val;  
 do {  
 cout << "\n1.Push\n2.Pop\n3.isEmpty\n4.isFull\n5.Display\n6.Peek\n7.Exit\nEnter choice: ";  
 cin >> choice;  
 switch (choice) {  
 case 1:  
 cout << "Enter value: ";  
 cin >> val;  
 s.push(val);  
 break;  
 case 2:  
 s.pop();  
 break;  
 case 3:  
 cout << (s.isEmpty() ? "Stack is empty\n" : "Stack is not empty\n");  
 break;  
 case 4:  
 cout << (s.isFull() ? "Stack is full\n" : "Stack is not full\n");  
 break;  
 case 5:  
 s.display();  
 break;  
 case 6:  
 s.peek();  
 break;  
 case 7:  
 cout << "Exiting...\n";  
 break;  
 default:  
 cout << "Invalid choice\n";  
 }  
 } while (choice != 7);  
 return 0;  
}

## 2. Reverse a string using stack

#include <iostream>  
#include <stack>  
using namespace std;  
  
string reverseString(string str) {  
 stack<char> s;  
 for (char ch : str)  
 s.push(ch);  
 string rev = "";  
 while (!s.empty()) {  
 rev += s.top();  
 s.pop();  
 }  
 return rev;  
}  
  
int main() {  
 string str = "DataStructure";  
 cout << "Original String: " << str << endl;  
 cout << "Reversed String: " << reverseString(str) << endl;  
 return 0;  
}

## 3. Check if expression has balanced parentheses

#include <iostream>  
#include <stack>  
using namespace std;  
  
bool isBalanced(string expr) {  
 stack<char> s;  
 for (char ch : expr) {  
 if (ch == '(' || ch == '{' || ch == '[')  
 s.push(ch);  
 else if (ch == ')' || ch == '}' || ch == ']') {  
 if (s.empty()) return false;  
 char top = s.top();  
 s.pop();  
 if ((ch == ')' && top != '(') ||  
 (ch == '}' && top != '{') ||  
 (ch == ']' && top != '['))  
 return false;  
 }  
 }  
 return s.empty();  
}  
  
int main() {  
 string expr;  
 cout << "Enter expression: ";  
 cin >> expr;  
 if (isBalanced(expr))  
 cout << "Balanced\n";  
 else  
 cout << "Not Balanced\n";  
 return 0;  
}

## 4. Convert Infix expression to Postfix

#include <iostream>  
#include <stack>  
using namespace std;  
  
int precedence(char c) {  
 if (c == '^') return 3;  
 else if (c == '\*' || c == '/') return 2;  
 else if (c == '+' || c == '-') return 1;  
 else return -1;  
}  
  
string infixToPostfix(string s) {  
 stack<char> st;  
 string result = "";  
 for (char c : s) {  
 if (isalnum(c))  
 result += c;  
 else if (c == '(')  
 st.push(c);  
 else if (c == ')') {  
 while (!st.empty() && st.top() != '(') {  
 result += st.top();  
 st.pop();  
 }  
 st.pop();  
 } else {  
 while (!st.empty() && precedence(st.top()) >= precedence(c)) {  
 result += st.top();  
 st.pop();  
 }  
 st.push(c);  
 }  
 }  
 while (!st.empty()) {  
 result += st.top();  
 st.pop();  
 }  
 return result;  
}  
  
int main() {  
 string exp;  
 cout << "Enter infix expression: ";  
 cin >> exp;  
 cout << "Postfix expression: " << infixToPostfix(exp) << endl;  
 return 0;  
}

## 5. Evaluate a Postfix expression

#include <iostream>  
#include <stack>  
using namespace std;  
  
int evaluatePostfix(string exp) {  
 stack<int> s;  
 for (char c : exp) {  
 if (isdigit(c))  
 s.push(c - '0');  
 else {  
 int val2 = s.top(); s.pop();  
 int val1 = s.top(); s.pop();  
 switch (c) {  
 case '+': s.push(val1 + val2); break;  
 case '-': s.push(val1 - val2); break;  
 case '\*': s.push(val1 \* val2); break;  
 case '/': s.push(val1 / val2); break;  
 }  
 }  
 }  
 return s.top();  
}  
  
int main() {  
 string exp;  
 cout << "Enter postfix expression: ";  
 cin >> exp;  
 cout << "Evaluation result: " << evaluatePostfix(exp) << endl;  
 return 0;  
}