BALANCE PARANTHESIS BY STACK

#include <iostream>

#include <stack>

#include <string>

bool isBalanced(const std::string& expression) {

std::stack<char> charStack;

for (size\_t i = 0; i < expression.size(); ++i) {

char c = expression[i];

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

// If the character is an opening parenthesis, push it onto the stack

charStack.push(c);

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

// If the character is a closing parenthesis

if (charStack.empty()) {

// If the stack is empty, there is no matching opening parenthesis

return false;

}

char top = charStack.top();

charStack.pop();

// Check if the top of the stack matches the current closing parenthesis

if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{')) {

// If there is a mismatch, the expression is not balanced

return false;

}

}

}

// If the stack is empty at the end, the expression is balanced

return charStack.empty();

}

int main() {

std::string expression1 = "{[()()]}";

std::string expression2 = "({[)]}";

std::string expression3 = "([])(){}";

std::string expression4 = "(((())))";

std::cout << std::boolalpha; // Print "true" or "false" instead of 1 or 0

std::cout << "Expression 1 is balanced: " << isBalanced(expression1) << std::endl; // true

std::cout << "Expression 2 is balanced: " << isBalanced(expression2) << std::endl; // false

std::cout << "Expression 3 is balanced: " << isBalanced(expression3) << std::endl; // true

std::cout << "Expression 4 is balanced: " << isBalanced(expression4) << std::endl; // true

return 0;

}

