(C++): Implement a stack-based parser in C++ to validate expressions containing

Below is a complete C++ program that implements a stack-based parser to check if an input string has balanced parentheses '(' and ')'. The program ignores other characters in the expression, focusing only on the parentheses as per the problem (validating expressions containing them). This is a classic use of a stack for matching nested structures, which relates to bottom-up parsing concepts where a stack simulates shift-reduce actions.

The logic:

- Use a stack to track opening parentheses '('.
- For each '(', push it onto the stack.
- For each ')', pop from the stack if it's not empty; otherwise, it's unbalanced.
- At the end, the stack must be empty for balance.
- This handles nesting and ensures proper matching, detecting errors like missing closers or extras.

```
#include <iostream>
#include <stack>
#include <string>

bool isBalanced(const std::string& expr) {
    std::stack<char> stk;
    for (char ch : expr) {
        if (ch == '(') {
            stk.push(ch);
        } else if (ch == ')') {
            if (stk.empty()) {
                return false;  // More closing than opening
            }
            stk.pop();
        }
        // Ignore other characters, as the focus is on ( and )
    }
    return stk.empty();  // True if all opened are closed
}

int main() {
    std::string input;
```

```
    std::cout << "Enter an expression: ";
    std::getline(std::cin, input);

    if (isBalanced(input)) {
        std::cout << "The parentheses are balanced." << std::endl;
    } else {
        std::cout << "The parentheses are not balanced." << std::endl;
    }

    // Example tests (for demonstration)
    std::cout << "Test '()': " << (isBalanced("()") ? "Balanced" : "Unbalanced") << std::endl;
    std::cout << "Test '(())': " << (isBalanced("(())") ? "Balanced" : "Unbalanced") << std::endl;
    std::cout << "Test '(()': " << (isBalanced("((") ? "Balanced" : "Unbalanced") << std::endl;
    std::cout << "Test ')(': " << (isBalanced(")(") ? "Balanced" : "Unbalanced") << std::endl;
    std::cout << "Test 'a+(b-c)': " << (isBalanced("a+(b-c)") ? "Balanced" : "Unbalanced") << std::endl;

    return 0;
}
```

- This implementation is efficient (O(n) time, O(n) space for the stack in the worst case of all opens). It demonstrates stack usage in parsing, similar to how shift-reduce parsers use stacks for bottom-up analysis. In a full compiler, this could be part of syntax validation for expressions, with error recovery (e.g., reporting mismatch positions) as an extension. The code is testable and robust, handling empty strings (balanced) and non-parenthesis characters.