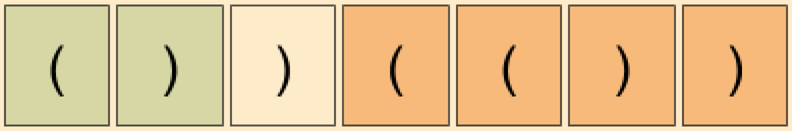
# GetAhead - Interview Practice 2

## Balanced Parentheses - Solution

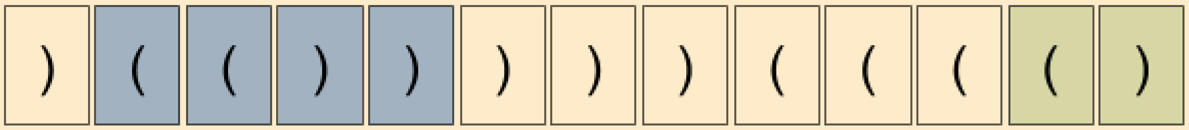
Given a string of parentheses, find the size of the longest contiguous substring of balanced parentheses. Parentheses are considered balanced when there is a valid closing parenthesis for an opening one.

**Example**:

In this string: `())(())`, the longest continuous set would be 4 characters long (the last 4 characters of the input):



For `)(())))((((()` , the max length would be 4:



### Solution

The following three solutions use a linear approach to scan the string exactly once, and make use of a Stack to keep track of the open parentheses that still need to be closed. The ability of the candidate to choose the appropriate data structure to solve a problem is one of the qualities that interviewers look for in a candidate.

#### JAVA

```java
// In an interview you can generally omit the import statements,
// however if you decide to use some library (like Stack in this example)
// it's a good idea to mention it to the Interviewer and ask if it's allowed.
import java.util.Stack;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class BalancedParentheses {
  public static int longestBalanced(String str) {
```

```java
  // Seed the stack with a "base" index of -1. The base holds the index
  // immediately before the current run of balanced parentheses.
  Stack<Integer> stack = new Stack<>();  // O(n) space
  stack.push(-1);
  int longest = 0;
  for (int i = 0; i < str.length(); i++) {  // O(n) time
    if (str.charAt(i) == '(') {
      // Remember the index of the opening parenthesis.
      stack.push(i);
    } else {
      // A closing parenthesis either matches and discards the last opening one,
      // or is unmatched and discards the base index.
      stack.pop();
      if (!stack.empty()) {
        // Compute the current length against the last open parenthesis or the
        // base (which occurs when the current chain is fully balanced), and
        // track the maximum of these.
        longest = Math.max(longest, i - stack.peek());
      } else {
        // This is the unmatched case, which means the current chain of balanced
        // parentheses has been broken. Reset the base index to our current index
        // to start tracking a new chain, starting from the next character.
        stack.push(i);
      }
    }
  }
  // Return the maximum.
  return longest;
}

// You don't necessarily have to write test code in an interview, but
// you are still expected to provide meaningful test cases and try some
// manually.
@Test
public static void testLongestBalanced() {
  assertEquals(0, longestBalanced(""));
  assertEquals(0, longestBalanced("("));
  assertEquals(0, longestBalanced(")"));
  assertEquals(2, longestBalanced("()("));
  assertEquals(2, longestBalanced("())"));
  assertEquals(4, longestBalanced("(())"));
  assertEquals(4, longestBalanced("()()"));
  assertEquals(6, longestBalanced("(()())"));
  assertEquals(4, longestBalanced("())(())"));
  assertEquals(4, longestBalanced(")(()))))(((()"));
  assertEquals(16, longestBalanced("(()()()(((()))))"));
}
```

```
}
```

**C++**

```cpp
// In an interview you can generally omit the include statements,
// however if you decide to use some library (like std::stack in this example)
// it's a good idea to mention it to the Interviewer and ask if it's allowed.
#include <string>
#include <cassert>
#include <stack>

int longest_balanced(std::string str) {
  // Seed the stack with a "base" index of -1. The base holds the index
  // immediately before the current run of balanced parentheses.
  std::stack<int> stack({-1});  // O(n) space
  int longest = 0;
  for (int i = 0; i < str.size(); i++) {  // O(n) time
    if (str[i] == '(') {
      // Remember the index of the opening parenthesis.
      stack.push(i);
    } else {
      // A closing parenthesis either matches and discards the last opening one,
      // or is unmatched and discards the base index.
      stack.pop();
      if (!stack.empty()) {
        // Compute the current length against the last open parenthesis or the
        // base (which occurs when the current chain is fully balanced), and
        // track the maximum of these.
        longest = std::max(longest, i - stack.top());
      } else {
        // This is the unmatched case, which means the current chain of balanced
        // parentheses has been broken. Reset the base index to our current index
        // to start tracking a new chain, starting from the next character.
        stack.push(i);
      }
    }
  }
  // Return the maximum.
  return longest;
}

// You don't necessarily have to write test code in an interview, but
// you are still expected to provide meaningful test cases and try some
// manually.
int main() {
  assert(longest_balanced("") == 0);
  assert(longest_balanced("(") == 0);
  assert(longest_balanced(")") == 0);
  assert(longest_balanced("()(") == 2);
  assert(longest_balanced("())") == 2);
  assert(longest_balanced("(())") == 4);
  assert(longest_balanced("()()") == 4);
  assert(longest_balanced("(()())") == 6);
  assert(longest_balanced("())(())") == 4);
  assert(longest_balanced(")(()))))(((()") == 4);
  assert(longest_balanced("(()())((((()))))") == 16);
}
```

### Python

```python
def longest_balanced(string):
  # Seed the stack with a "base" index of -1. The base holds the index
  # immediately before the current run of balanced parentheses.
  stack = [-1]  # O(n) space.
  longest = 0
  for i, char in enumerate(string):
    if char == '(':
      # Remember the index of the opening parenthesis.
      stack.append(i)
    elif char == ')':
      # A closing parenthesis either matches and discards the last opening one,
      # or is unmatched and discards the base index.
      stack.pop()
      if stack:
        # Compute the current length against the last open parenthesis or the
        # base (which occurs when the current chain is fully balanced), and
        # track the maximum of these.
        longest = max(longest, i - stack[-1])
      else:
        # This is the unmatched case, which means the current chain of balanced
        # parentheses has been broken. Reset the base index to our current index
        # to start tracking a new chain, starting from the next character.
        stack.append(i)
  # Return the maximum.
  return longest

# You don't necessarily have to write test code in an interview, but
# you are still expected to provide meaningful test cases and try some
# manually.
if __name__ == '__main__':
  assert longest_balanced('') == 0
  assert longest_balanced('(') == 0
  assert longest_balanced(')') == 0
  assert longest_balanced('()(') == 2
  assert longest_balanced('())') == 2
  assert longest_balanced('(())') == 4
  assert longest_balanced('()()') == 4
  assert longest_balanced('(()())') == 6
  assert longest_balanced('())(())') == 4
  assert longest_balanced(')(())))(((()') == 4
  assert longest_balanced('(()())((((()))))') == 16
```