CSC236 tutorial exercises, Week #7
best before Friday afternoon

1. For a given string s, we'll say that s' is a **top-level parenthesized substring** ('tlps') of s iff the following conditions are met:

   i. s' is a substring of s

   ii. s' is of the form (q), where q is a (possibly empty) string with balanced parentheses

   iii. there does not exist any longer string, s'', such that s' is contained within s'' and s'' satisfies the above 2 conditions

   For example, the string $s =$ '((hello))hi((a)(b)(c))' has two tlps's: '((hello))' and '((a)(b)(c))'. On the other hand

   - '(a)' is not a tlps, because it is contained within the tlps '((a)(b)(c))'
   - '((hello))hi' is not a tlps, because it is not enclosed in parentheses
   - s itself is not a tlps because the parentheses in '(hello))hi((a)(b)(c)' are not balanced

```
def extract_tlps(s):
    """Pre: s is a string, and its parentheses are balanced.
    Post: Return a list of all the tlps's in s

    >>> extract_tlps('sorry (not sorry)')
    ['(not sorry)']
    >>> extract_tlps('((hello))hi((a)(b)(c))')
    ['((hello))', '((a)(b)(c))']
    """
    R = []
    l = 0
    par = ''
    i = 0
    while i < len(s):
        c = s[i]
        if c == '(':
            l += 1
        if l > 0:
            par += c
        if c == ')':
            l -= 1
            if l == 0:
                R.append(par)
                par = ''
        i += 1
    return R
```

(a) Devise a loop invariant for this function which is sufficient to prove partial correctness. Do not attempt to prove the invariant.[1]

(b) Assume the invariant you wrote in part (a) is true, and use it to prove the partial correctness of `extract_tlps` (i.e. if `extract_tlps(s)` terminates, then the postcondition is satisfied).

(c) In the above parts, you may have found that a simple invariant was sufficient to prove partial correctness. However, proving that invariant directly by induction may prove very difficult. In this case, we need to strengthen the induction, by adding additional invariants which help us prove the main one. Here are two useful examples:

i. l_j is the 'left surplus' of par_j, i.e. the count of left parentheses in par_j minus the count of right parentheses in par_j

ii. l_j is the left surplus of $s[:i_j]$.

Use induction to prove these invariants.

Hint: You may use assume that any prefix of a string with balanced parentheses has at least as many left parentheses as right parentheses. (We proved essentially this fact in lecture when talking about structural induction.)

(d) Is the invariant from part (c) enough to prove your original invariant from part (a)? Brainstorm additional invariants which could be used to complete the proof. You do not need to prove your final set of invariants (the full proof would be fairly long), but you should have some idea of what the overall structure of the proof would look like.

2. Consider the function `bitcount` defined below:

```
1 def bitcount(n):
2     """Pre: n is a positive int.
3     Post: return the number of digits in the binary representation of n
4     """
5     i = 1
6     while n > 1:
7         n = n//2
8         i += 1
9     return i
```

Prove that `bitcount` is partially correct with respect to the specification in the docstring. You will need to prove an appropriate loop invariant, then show that it implies the postcondition given that the loop exits.

You may assume that for $n \in \mathbb{N}^+$, n has k bits iff 2^{k-1} is the largest power of 2 which is less than or equal to n. Or, equivalently, $k = \lfloor \log n \rfloor + 1$.

[1] Even though you don't need to prove it, your invariant should be *true*. For example, "R_j contains all tlps's in s" or "$i_j = i_j + 1$" are not appropriate invariants, even though either would technically be sufficient for proving partial correctness.