COMP 211: Lab 7
Fall, 2014

To get the code for this lab, go to the Assignments page and save all the files and `lab07-*.c0` to your COMP211 directory. You can load them by typing

```
% coin -d lab07-list.c0 lab07-stack.c0 lab07-stack-client.c0
```

1 Stacks

Task 1.1 Your task is to implement stacks using lists in `lab07-stack.c0`. The implementation of lists is in `lab07-list.c0`. The analogous code for queues is on the lectures page, under Lecture 18.

- You need to fill in the `typedef` for `stack` at the top of `lab07-stack.c0`. You will need to define a new struct for representing stacks.
- You should define a data-structure-invariant-function `is_stack` that describes what a valid stack looks like. You can use the function `has_cycle` from `lab07-list.c0`, which you will implement in the next part of the lab (it always returns false for now).
- Each of the functions you implement should require and ensure the data structure invariant.
- The interface specifies that all of the stack functions run in constant time. However, for this task, write `stack_length` in such a way that its running time is proportional to the size of the stack (you'll fix this below).
- You can use the `test()` function in `lab07-stack-client.c0` to test.

Task 1.2 Implement `stack_length` in constant time. You will likely need to change your `struct` representation of stacks.

2 Cycle Detection

Next, implement `has_cycle` in `lab07-list.c0`.

The algorithm you should use is called "the tortoise and the hare." The idea is this: the tortoise walks through the list, one element at a time. The hare starts one node ahead of the tortoise, and moves twice as fast: the hare takes two steps for every step the tortoise takes. The hare will reach

the end of the list iff there is no cycle. The hare (which started off ahead of the tortoise) will eventually be at the same node as the tortoise iff there is a cycle.

Hint: represent the tortoise and the hare as two pointers into the list.