15-122: Principles of Imperative Computation

Recitation 11 — Josh Zimmerman

Circularity checking

1) Is this a correct implementation? Is the hare capable of "skipping over" the tortise when approaching from behind? If so, what is the appropriate fix?

This is a correct implementation: the key observation is that the hare approaches the tortise from behind, and the distance between them only gets smaller by 1 on every iteration through the loop:

```
_H_____T__________
___H____T_________
_____H___T________
_______H__T_______
_________H_T______
___________H_T____
_____________HT___
_______________!__
```

2) How many times is a pointer accessed within the loop? How do we know each access is safe? What happens if `h->next->next` is `NULL` at the beginning of a loop?

There are three pointer we dereference: `t` on line 10, `h` on line 11, and `h->next`, again on line 11. If `h->next->next` is `NULL` at the start of the loop, then `h` will be `NULL` when the loop body terminates. This means that it would be unsafe to dereference `h`, but we never will: on the next run of the loop we will notice on line 9 that `h` is `NULL` and the function will return `false`.

3) The check `t == NULL` on line 8 is unnecessary. First come up with a rough operational reason why this is the case, then state this reason in terms of a loop invariant involving `is_segment`.

Operationally, the tortise is only ever treading along ground that the hare has already covered. Because our `is_segment` function doesn't allow either endpoint of the segment to be `NULL`, it's a bit difficult to write a good loop invariant for the function as written. One option would be to write a variant of `is_segment`, another is to rewrite the function slightly. The lecture 11 notes have more discussion of this pont.

```
1  bool is_circular(list* l)
2  {
3    if (l == NULL) return false;
4    list* t = l;  // tortoise
5    list* hprev = l; // one prior to the hare
6    while (t != hprev->next)
7    //@loop_invariant is_segment(t, hprev);
8    {
9      list* h = hprev->next;
10     if (t == NULL) return false;
11     if (h == NULL || h->next == NULL) return false;
12     t = t->next;
13     hprev = h->next;
14   }
15   return true;
16 }
```