15-122: Principles of Imperative Computation

QuickCheck 4 Solutions

Today, we'll do the QuickCheck at the beginning of recitation. You will have ten minutes to do this. Your TA will go over answers at the end. Then follow your TA's instructions on whether or not to hand it in

Name:

Andrew ID:

Section (circle one): A B C D E F G H

Simplicio and *Sagredo* (you) are a couple of 122 students. They are trying to complete an exercise, but are a bit stuck. Help them out!

After looking at linked lists in lecture, *Simplicio* believes that he has finally found something that you can do using a `while` loop but not a `for` loop. Show him that you can, indeed, write a `for` loop to find the last element of a given linked list. (Just for this question, assume you have a `is_valid_list()` function, which says the list is not null initially)

```
1  typedef struct list_node list;
2
3  struct list_node {
4    int data;
5    list* next;
6  };
7
8  int find_last(list* L)
9  //@requires is_valid_list(L);
10 //@ensures is_valid_list(L);
11 {
12
13       for(                  ;                  ;                  )
14       {
15
16
17       }
18
19 }
```

Solution:

```
1  int find_last(list* L)
```

```
2 //@requires is_valid_list(L);
3 //@ensures is_valid_list(L);
4 {
5       for(list* p = L; p->next != NULL; p = p->next)
6       {
7
8       }
9       return p->data;
10 }
```

(*Salviati*: Psst, Line 18 above is blank for a reason.....)

Salviati: Also, just because you **can** write code this way, doesn't mean you should. A `while` loop makes more intuitive sense here.

Simplicio just can't get the hang of pointers. He used to think that they were some kind of aliases and got really, terribly confused when *Salviati* told him that that wasn't true. Now *Simplicio* want you, *Sagredo*, his ever helpful classmate, to tell him the output of the following, so he can clear his misconceptions.

```
1 int pointer_help{
2       list* p = alloc(struct list_node);
3       p->data = 2;
4       list* q = p;
5       q->data = 3;
6       printint(p->data);
7       list* r = alloc(struct list_node);
8       r->data = 5;
9       p = r;
10      printint(p->data);
11      printint(q->data);
12      r = q;
13      printint(p->data);
14      printint(q->data);
15      printint(r->data);
16      return 0;
17 }
```

3
5
3
5
3
3