15-122: Principles of Imperative Computation — Spring 2018

Recitation 5: A queue_t Interface — 8 February

A Wild struct Appears

Suppose we have the following definitions:

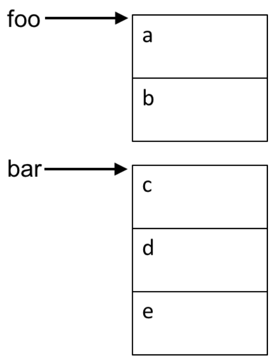
```
struct X {
  int a;
  struct Y* b;
};

struct Y {
  int* c;
  int d;
  struct X* e;
};

struct X* foo = alloc(struct X);
struct Y* bar = alloc(struct Y);

foo->b = bar;
bar->e = foo;

bar->e->a = 15;
foo->b->c = alloc(int);
*(bar->c) = foo->a * 8 + 2;
foo->b->d = 1000 * foo->a + *(foo->b->c);
```



Checkpoint 0

Fill out the table above. What's the value of bar->d? (For your own sanity, draw a picture!)

Stack and Queue Interfaces

In lecture we discussed four functions exposed by the stack interface:

- stack_new: Creates and returns a new stack
- stack_empty: Given a stack, returns true if it is empty, else false
- push: Given a stack and a string, puts the string on the top of the stack
- pop: Given a stack, removes and returns the string on the top of the stack

Similarly, we discussed four functions exposed by the queue interface:

- queue_new: Creates and returns a new queue
- queue_empty: Given a queue, returns true if it is empty, else false
- enq: Given a queue and a string, puts the string at the end of the queue
- deq: Given a queue, removes and returns the string at the beginning of the queue

Checkpoint 1

Write a function to reverse a queue using only functions from the stack and queue interfaces.

```
void reverse(queue_t Q) {
______________________________________________  // Hint: Allocate a
________________________________________  // temporary data structure
  while (_____________________________________________________) {
    ___________________________________________________________
    ___________________________________________________________
  }
  while (_____________________________________________________) {
    ___________________________________________________________
    ___________________________________________________________
  }
}
```

Checkpoint 2

Write a *recursive* function to count the size of a stack. You may not destroy the stack in the process — the stack's elements (and order) must be the same before and after calling this function.

```
int size(stack_t S) {
  ______________________________________________________________
  ______________________________________________________________
  ______________________________________________________________
  ______________________________________________________________
  ______________________________________________________________
  ______________________________________________________________
}
```

Checkpoint 3

Why couldn't this stack size implementation be used in contracts in C0? Hint: Contracts in C0 cannot have side effects.

____