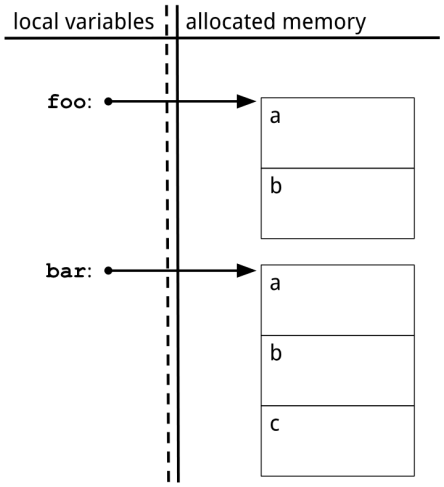
Recitation 5: A `queue_t` Interface Thursday February 13th

A wild `struct` appears

Suppose we have the following in a file:

```
1  struct X {
2    int a;
3    struct Y* b; };
4
5  struct Y {
6    int* a;
7    int b;
8    struct X* c; };
9
10 int main() {
11   struct X* foo = alloc(struct X);
12   struct Y* bar = alloc(struct Y);
13
14   foo->b = bar;
15   bar->c = foo;
16   bar->c->a = 15;
17   foo->b->a = alloc(int);
18   *(bar->a) = foo->a * 8 + 2;
19   foo->b->b = 1000 * foo->a + *(foo->b->a);
20
21   return 0;
22 }
```



Checkpoint 0

Fill out the state of the memory. What's the value of `bar->b`? (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, and false otherwise
- `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, and false otherwise
- `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.

```
1  void reverse(queue_t Q)
2  //@requires _____________________________________________________________;
3  {
4  _____________________________________  // create temp data structure
5    while (_____________________________________________________________) {
6      ___________________________________________________________________
7    }
8    while (_____________________________________________________________) {
9      ___________________________________________________________________
10   }
11 }
```

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. Assume the stack contains elements of type **string**.

```
int size(stack_t S)
//@requires _____________________________________________________________;
{
  ___________________________________________________________________
  ___________________________________________________________________
  ___________________________________________________________________
  ___________________________________________________________________
  ___________________________________________________________________
}
```

Checkpoint 3

Why couldn't this function be used in contracts in C0? Hint: Contracts in C0 can't have side effects.