15-122: Principles of Imperative Computation

Recitation 9

Josh Zimmerman, Arjun Hans

Stacks and Queues

Here are some implementations of the functions we discussed in recitation. Other implementations are possible.

```
1 // Create only queues in the function
2 int queue_size(queue Q)
3 {
4   queue R = queue_new();
5   int i = 0;
6   while (!queue_empty(Q)) {
7     enq(R, deq(Q));
8     i++;
9   }
10  while (!queue_empty(R)) {
11    enq(Q, deq(R));
12  }
13  return i;
14 }
```

```
1 // Create only stacks in the function
2 int stack_size_1(stack S)
3 {
4   stack R = stack_new();
5   int i = 0;
6   while (!stack_empty(S)) {
7     push(R, pop(S));
8     i++;
9   }
10  while (!stack_empty(R)) {
11    push(S, pop(R));
12  }
13  return i;
14 }
```

```
1 // Recursive. Don't allocate any other data structures
2 int stack_size_2(stack S)
3 {
4   if (stack_empty(S)) return 0;
5   string x = pop(S);
6   int i = 1 + stack_size_2(S);
7   push(S, x);
8   return i;
9 }
```

```
1 // Create only queues in the function
2 void stack_reverse_1(stack S)
3 {
4   queue R = queue_new();
5   while (!stack_empty(S)) {
6     enq(R, pop(S));
7   }
8   while (!queue_empty(R)) {
9     push(S, deq(R));
10  }
11 }
```

```
1 // Create only stacks in the function
2 void stack_reverse_2(stack S)
3 {
4   stack R1 = stack_new();
5   stack R2 = stack_new();
6   while (!stack_empty(S)) {
7     push(R1, pop(S));
8   }
9   while (!stack_empty(R1)) {
10    push(R2, pop(R1));
11  }
12  while (!stack_empty(R2)) {
13    push(S, pop(R2));
14  }
15 }
```

```
1 // Recursive. Only allocate the one stack you return
2 stack stack_copy(stack S)
3 {
4   if (stack_empty(S)) return stack_new();
5   string x = pop(S);
6   stack R = stack_copy(S);
7   push(R, x);
8   return R;
9 }
```

```
1 // Only allocate the one queue you return
2 queue queue_copy(queue Q)
3 {
4   queue R = queue_new();
5   int i = 0;
6   while (!queue_empty(Q)) {
7     i++;
8     enq(R, deq(Q));
9   }
10
11  while (i > 0) {
12    i--;
13    //@assert !queue_empty(R);
14    string x = deq(R);
15    enq(Q, x);
16    enq(R, x);
17  }
18
19  return R;
20 }
```