15-122: Principles of Imperative Computation

Lab 2: A Reversal of Fortune

Shyam Raghavan, Tom Cortina

Setup: Copy the lab code from our public directory to your private directory:

```
% cd private/15122
% cp -R /afs/andrew/course/15/122/misc/lab-integers .
% cd lab-integers
```

You should write your code in a file, `reverse.c0`, in the directory `lab-integers`.

Grading: You will work through (1.a) and (1.b) as a group. Finish (2.a) for lab credit, and (2.b) for potentially more credit. Show your TA that you've passed the tests when you complete these parts. You can work on these problems with your neighbors and compare notes and solutions!

In this lab, you'll be implementing two distinct ways of reversing a seven-digit decimal number - one using mathematical operations (or bitwise operations, if you'd like!) and one using arrays. There are multiple ways to do each!

Reasoning about reversing a decimal number

For these two tasks, you'll need to use a loop to manipulate integers. Your loop should include loop invariants that allow you to prove the correctness and safety of the function.

(1.a) Our first task will be to write loop invariants for an algorithm that reverses the digits in a seven-digit decimal number (a number with fewer digits will be treated as having leading zeros) using mathematical operations. Although there is more than one way to do this, we've provided the steps of a suggested algorithm run on three examples below, which shows the state of each of the different variables at iteration `i` in the loop:

i	x	y
0	1234567	0
1	123456	7
2	12345	76
3	1234	765
4	123	7654
5	12	76543
6	1	765432
7	0	7654321

i	x	y
0	15122	0
1	1512	2
2	151	22
3	15	221
4	1	2215
5	0	22151
6	0	221510
7	0	2215100

i	x	y
0	2400000	0
1	240000	0
2	24000	0
3	2400	0
4	240	0
5	24	0
6	2	4
7	0	42

Can you suggest a couple of loop invariants for the algorithm above? **Hint:** you may want to use the `POW` specification from lecture. What can you say about `POW(10,i)`?

`//@loop_invariant` _____

`//@loop_invariant` _____

`//@loop_invariant` _____

Remember that if you use `POW`, you need your loop invariants to also ensure that the exponent will always be nonnegative.

(1.b) Next, we'll write loop invariants for an algorithm that reverses seven-digit numbers using arrays (note that this algorithm uses two sequential loops, denoted by `i` followed by `j`). Again, we provide steps of an algorithm that you can use when implementing the code in (2.b):

i	x	A						
0	1234567	0	0	0	0	0	0	0
1	123456	7	0	0	0	0	0	0
2	12345	7	6	0	0	0	0	0
3	1234	7	6	5	0	0	0	0
4	123	7	6	5	4	0	0	0
5	12	7	6	5	4	3	0	0
6	1	7	6	5	4	3	2	0
7	0	7	6	5	4	3	2	1

j	y	A						
0	0	7	6	5	4	3	2	1
1	7	7	6	5	4	3	2	1
2	76	7	6	5	4	3	2	1
3	765	7	6	5	4	3	2	1
4	7654	7	6	5	4	3	2	1
5	76543	7	6	5	4	3	2	1
6	765432	7	6	5	4	3	2	1
7	7654321	7	6	5	4	3	2	1

Again, let's suggest a couple of loop invariants for the second (right) loop above. **Hint:** make sure you're ensuring the safety of the array accesses!

`//@loop_invariant` _____

`//@loop_invariant` _____

Implementing the algorithms

(2.a) Now you have two good sets of loop invariants: in `reverse.c0`, use the algorithm from (1.a) to implement a function that reverses the decimal digits in a seven-digit nonnegative number using only mathematic and bitwise operators, and call it `reverse_math` (we've provided a skeleton function for you). You shouldn't have to use `POW` outside of contracts. **Treat a number with fewer than seven digits as if it has leading zeroes.**

```
% coin -d reverse.c0
-->> reverse_math(7654321);
1234567 (int)
-->> reverse_math(1512200);
22151 (int)
-->> reverse_math(42);
2400000 (int)
```

You can test your code by running `cc0 -d -x reverse.c0 test-math-rev.c0`

(2.b) In `reverse.c0`, write a function `reverse_array` that reverses all the decimal digits of a seven-digit nonnegative number using arrays and the algorithm from (1.b). Be sure to use the loop invariants you wrote above!

```
% coin -d reverse.c0
-->> reverse_array(7654321);
1234567 (int)
-->> reverse_array(1512200);
22151 (int)
-->> reverse_array(42);
2400000 (int)
```

You can test your code by running `cc0 -d -x reverse.c0 test-array-rev.c0`