15-122: Principles of Imperative Computation

Lab 2: A Reversal of Fortune

Tom Cortina, Nivedita Chopra

Two's complement

Because C0's `int` type only represents integers in the range $[-2^{31}, 2^{31})$, addition and multiplication are defined in terms of modular arithmetic. As a result, adding two positive numbers may give you a negative number!

Checkpoint 0

What assertion would you need to write to ensure that an addition would give a result without overflowing (in other words, to ensure that the result you get in C0 is the same as the result you get with true integer arithmetic).

```
1 int safe_add(int a, int b)
2 /*@requires
3
4
5
6
7 @*/
8 { return a + b; }
```

What about multiplication? For the sake of simplicity, you can assume both numbers are non-negative.

```
1 int safe_mult(int a, int b)
2 /*@requires a >= 0 && b >= 0 &&
3
4
5
6
7 @*/
8 { return a * b; }
```

Checkpoint 1

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`.

Manipulating integers with a loop

For this task, you'll need to use a loop to manipulate integers. We can identify two ways of manipulating integers in C0:

- The mathematical operations of multiplication (`a * b`), division (`a / b`), modulo (`a % b`) addition (`a + b`), subtraction (`a - b`), and negation (`-a`).
- The bitwise operations bitwise-and (`a & b`), bitwise-or (`a | b`), bitwise-xor (`a ^ b`), bitwise negation (`~a`), left shift (`a << b`) and right-shift (`a >> b`).

We don't always think about these operations as distinct categories! Sometimes, for instance, we think about `a << b` as the mathematical operation $a \times 2^b$. But for this assignment we will make the distinction.

(2.a) Our task will be to reverse the digits in a seven-digit decimal number (a number with fewer digits will be treated as having leading zeros). There's more than one way to do this! The three examples below show one way of reversing a number using a loop on the variable `i`:

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` _____

`//@loop_invariant` _____

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