15-122: Principles of Imperative Computation, Fall 2014
Lab 2: Ints and Arrays

Tom Cortina(tcortina@cs)

Monday, September 8, 2014

SETUP: In your `15122` subdirectory in your `private` directory, create a new directory named `lab2`. Store all of your functions below in the `lab2` directory.

1 Ints

Exercise 1. Write a C0 function `reverse_int` in the file `reverse_int.c0` that has one parameter `i` of type `int` and returns an `int` that is the bitwise reverse of the integer parameter. For example, `reverse_int(0x89ABCDEF)` should return `0xF7B3D591`. Use a loop in your solution, and include a loop invariant for the loop. Use `coin` to test your function carefully. Be sure to run `coin` with `-lutil` and `-d`.

Once you are done, add a comment to your file that answers the following statement: Explain why we cannot include the following postcondition to the function if we are running using `-d`:

```
//@ensures i == reverse_int(\result);
```

Exercise 2. Write a C0 function `reverse_num` in the file `reverse_num.c0` that has one parameter `n` of type `int` representing a 7 digit positive decimal integer. This function should return the decimal reversal of the given integer using a loop. Programming hint: Think about the result you get for a positive integer modulo 10.

Here are some examples:

```
--> reverse_num(1357246);
6427531 (int)
--> reverse_num(15122);
2215100 (int)
--> reverse_num(42);
2400000 (int)
```

Include a suitable precondition for your function. Use `coin` to test your function carefully. Be sure to run `coin` with `-d`.

2 Arrays

Exercise 3. Write a C0 function `squares_array` in the file `array_ops.c0` that has one *non-negative* `int` parameter representing the number of elements for an array. This function should allocate an array that can hold the given number of integers, initialize the array so that each integer at index i is equal to i^2 for all valid indices, and return a reference to this array. Include a suitable precondition and postcondition for your function. Also write a suitable loop invariant for the loop in your function. Use `coin` to test your function carefully. Be sure to run `coin` with `-d`.

Exercise 4. Write a C0 function `sum_array` in the same file `array_ops.c0` that has two parameters: a reference to an array `A` of integers and an `int n` representing the total number of elements of the array. The function should return the sum of the integers in the array `A`. Your function should use a loop to compute the answer by adding each element to an accumulator one at a time. Include suitable preconditions for your function. Use contracts to ensure the safety of all array accesses. Use `coin` to test your function carefully. Be sure to run `coin` with `-d`.

Sample execution (your array location will likely vary):

```
--> int[] X = squares_array(10);
X is 0x14A9010 (int[] with 10 elements)
--> sum_array(X, 10);
285 (int)
```

Add a postcondition that verifies your sum. (HINT: There is a closed form for the sum of the squares 0 + 1 + 4 + 9 + 16 + ...). Test again.

Once you are done, note that overflow will occur if the sum of squares exceeds $2^{31} - 1$. Test your function to find the minimum square that causes the total sum to overflow and then add an additional precondition so overflow does not occur. That is, limit the maximum size of the array. Test again to make sure everything still works correctly.

Exercise 5. Challenge: Rewrite `sum_array` so that it uses recursion to compute the sum of the squares in the array rather than iteration (i.e. a loop). HINT: You will need to break this problem down into a subproblem that computes the sum of the numbers in the array starting at index `i`.

3 Wrap-up

Show one of your teaching assistants your work before you leave in order to get credit for the lab.

MAKE SURE YOU LOG OUT BEFORE YOU LEAVE!