CSE 331 Software Design and Implementation

Homework 2: ***Reasoning About Loops***
Due: Friday April 12 @ 11pm

Contents:

Introduction

In this part of the assignment you will practice reasoning about code to prove the correctness of loops. Write your answers to problems 1-4 in a **text** file named `hw2/answers/hw2_answers.txt`.

Problems

Problem 1: Computing the Maximum Value in a List

(Warmup) The lecture reading gives an example of an algorithm to find the largest value in a non-empty list of integers `items[0..size-1]`. In that code, the loop has the following invariant:

```
max = largest value in items[0..k-1]
```

Suppose we decide to use the following slightly different invariant instead:

```
max = largest value in items[0..k]
```

(The difference is that the upper bound of the array section is `k` instead of `k-1`.)

Rework the code in the example to use this new invariant instead of the original one and show that the modified code is correct. Insert or modify assertions and code as needed.

After solving this problem, give a brief description of how this change to the invariant affected the algorithm. What were the major changes? Did this change make the code easier or harder to write or prove compared to the original version? Why? (You should be

able to keep your answers brief and to the point.)

Problem 2: Calculating x^y

Given two non-negative integers x and y, we can calculate the exponential function x^y (x raised to the power y) using repeated multiplications by x a total of y times. The following code is alleged to compute x^y much more quickly (it actually takes time proportional to lg y).

```
@requires x >= 0 && y >= 0
int expt(int x, int y) {
  int z = 1;
  while (y != 0) {
    if (y % 2 == 0) {
      y = y/2;
      x = x*x;
    } else {
      z = z*x;
      y = y-1;
    }
  }
  return z;
}
```

Give a proof that this code is correct. A suitable precondition for the function would be $x==x_{pre}$ && $y==y_{pre}$ && $x_{pre}?0$ && $y_{pre}?0$, and an appropriate postcondition would be that the returned value $z==x_{pre}\text{^}y_{pre}$. (We define 0^0 to be 1.) You will need to develop a loop invariant and pre- and post-conditions for the statements inside the loop, and verify that the code has the necessary properties to ensure that these assertions hold. You are not required to give pre- and post-conditions for each individual assignment statement inside the `if` if these are not needed to clearly show the code is correct, but you should include whatever is needed so that the grader can follow your proof and see that it is valid.

Problem 3: Reordering an Array

We are given an integer array `a` containing `n` elements and would like to develop an algorithm to rearrange the array elements as follows and prove that the algorithm is correct. If the original elements of the array are A[0], A[1], ..., A[n-1], we would like to rearrange the array so that all of the elements less than or equal to the original value in A[0] are at the beginning of the array, followed by the value originally stored in A[0], followed by all the elements in the original array that are larger than A[0]. In pictures, the postcondition we want is the following:

```
  0                                  n
 +-------------+----+---------------+
a|  <= A[0]    |A[0]|      >A[0]    |
 +-------------+----+---------------+
```

The operation `swap(a[i],a[j])` can be used to interchange any pair of elements in the array, and this is the only operation that can be used to modify the contents of the array. (As a result of this restriction, the array will be a permutation of its original contents, so you do not need to prove this.) Your code should run in linear ($O(n)$) time.

You should develop a suitable loop invariant, then write code to partition the array using that invariant and prove that when the code terminates the postcondition has been established. You do not need to provide assertions for every trivial statement in the code, but there should be sufficient annotations so that the correctness of your code is obvious. You do not need to write a complete procedure, function, or method, just the necessary loop and supporting statements, including any necessary initialization and final statements before and after the loop.

Hints: As you are partitioning the values in the array, you might find it simpler if you are not constantly moving the original value from `A[0]` into new locations. Try different invariants and see which one makes things simplest. Also remember that you can include additional statements before and after the main loop if useful to establish the loop invariant or the postcondition.

Problem 4: Selection Sort

Give an implementation of the algorithm *selection sort* and a proof of its correctness. The algorithm should sort an array `a` containing `n` integer values. The precondition is that the array contains `n` integer values in some unknown order. The postcondition is that the array holds a permutation of the original values such that `a[0] ? a[1] ? ... ? a[n-1]`. As in the previous problem you should use the operation `swap(a[i],a[j])` to interchange array elements when needed.

Selection sort proceeds as follows. First we find the smallest element in the original array and swap it with the original value in `a[0]`. (If `a[0]` is the smallest element in the original array it is fine to swap it with itself to avoid having additional special cases in the code.) Next we find the smallest element in the remaining part of the array beginning at `a[1]` and swap it with `a[1]`. Then we find the smallest element in the remaining part of the array starting at `a[2]` and move it to the front of that part of the array. This search-and-swap operation continues on the successively smaller remaining parts of the array until all elements are sorted.

As with the previous problem, you should develop suitable invariants for the nested loops needed to perform the sort, then write the code and annotate it with appropriate assertions so that it is clear that your code is correct and that when it terminates the array is sorted.

HW3, Problem 0: Polynomial Long Division

Complete Problem 0 on Homework 3 by the deadline for Homework 2. This problem involves writing several algorithms for Homework 3, which we will review and return to you before you are required to implement them in Homework 3. For ease of grading, submit this as a separate **text** file under **hw2** (not hw3) named hw2/answers/hw3_problem0.txt.

Collaboration

Please answer the following questions in a file named `collaboration.txt` in your `hw2/answers/` directory.

The standard collaboration policy applies to this homework.

State whether or not you collaborated with other students. If you did collaborate with other students, state their names and a brief description of how you collaborated.

Reflection

Please answer the following questions in a file named `reflection.txt` in your `hw2/answers/` directory. Answer briefly, but in enough detail to help you improve your own practice via introspection and to enable the course staff to improve CSE 331 in the future.

a. In retrospect, what could you have done better to reduce the time you spent solving

this homework?

b. What could the CSE 331 staff have done better to improve your learning experience in this homework?

c. What do you know now that you wish you had known before beginning the homework?

Time Spent

Tell us how long you spent on this homework via this catalyst survey: https://catalyst.uw.edu/webq/survey/mernst/198072.

Turnin

You should add and commit the following files to SVN:

- `hw2/answers/hw2_answers.txt`
- `hw2/answers/hw3_problem0.txt`
- `hw2/answers/reflection.txt`
- `hw2/answers/collaboration.txt`

Don't forget to run `ant validate` to ensure you have submitted all files correctly.

Hints

When trying to come up with a loop invariant for prewritten code, it often helps to trace through the execution of the code on paper. Choose a few different starting values of variables defined outside the block of code (such as method arguments), and write down the values of all the variables used in the loop for each iteration.

Errata

None yet.

Q & A

This section will list clarifications and answers to common questions about homeworks. We'll try to keep it as up-to-date as possible, so this should be the first place to look (after carefully rereading the homework handout and the specifications) when you have a problem.