A GIRL HAS NO NAME
Computer Science 81
Homework 5
2/21/17

1, 2, 3, 4, 5

All proofs must be typed. We don't want symbolic Hoare-Logic proofs, but rather readable proofs in mathematical English as one would use for a math class or the Algorithms class (or the background survey from the start of this semester). Try to be as clear as possible, use good grammar and complete sentences, and proofread your answers before you turn them in.

1. [19 points] Consider the following code for finding the maximum of a function f on inputs $0 \ldots (N-1)$.

$\{N \geq 1\}$

```
j=1
m = f(0)
while (j != N):
    if m > f(j):
        j = j + 1
    else:
        m = f(j)
        j = j + 1
```

$\{m = \max_{k \in 0 \ldots N-1} f(k)\}$

Your job is to convincingly argue it works as intended (terminates with the correct answer.)

A. [5 points] Propose a suitable loop invariant I for this code.

B. [9 points] Justify your choice of I by briefly explaining (using complete sentences):

   a. Why your invariant I is true right before the while loop begins;

   b. Why the loop body preserves the invariant (that is, why I is true at the end of the loop body, when the "if" is done, assuming I is true at the start of the loop body);

   c. Why the invariant I after the loop (plus the fact that the loop terminated) proves the desired postcondition.

C. [5 points] Specify a loop variant that could be used to prove termination. (No justification is required.)

■

2. [35 points] The following code computes a mod d (where $a \geq 0$ and $d > 0$) via repeated subtraction, and puts the final result in r. Instead of just repeatedly subtracting d, it tries to speed things up (e.g., when a is much larger than d) by subtracting larger and larger multiples of d where possible.

$\{a \geq 0 \wedge d > 0\}$

```
r = a
while (r >= d):
    g = d
    while (r >= g):
        r = r - g
        g = g + g
```

$\{r = (a \bmod d)\}$

Your job is to convincingly argue that this code works as intended (terminates with the correct answer.) If you're not sure how/why it works, simulating the code by hand (on various inputs) is an excellent way to gather evidence and hypotheses for useful loop invariants.

A. [10 points] Propose a suitable invariant I_1 for the outer while loop.

B. [10 points] Propose a suitable invariant I_2 for the inner while loop. (Hint: it probably should mention g.)

C. [10 points] Briefly explain (using words and complete sentences)

   a. Why is your invariant I_1 is true right before the outer loop begins;

   b. Why is your invariant I_2 is true right before the inner loop begins;

   c. Why does the code in the inner loop preserves the invariant I_2 (that is, why I_2 is still true at the end of the inner loop body);

   d. Why does the invariant I_2 after the inner loop (plus the fact that the inner loop terminated) proves that I_1 still holds (i.e., that the outer loop preserves the invariant I_1);

   e. Why does the invariant I_1 after the outer loop (plus the fact that the outer loop terminated) prove the desired postcondition.

D. [5 points] Give variants for both loops. (No justification is required.)

■

3. [35 points] Finally, here is some code that sets z to be a^b, where a and b are nonnegative integers. For efficiency, it uses repeated squaring. (Recall that % is the standard way of writing the "modulo" or "remainder" operator in code.)

```
{a > 0 ∧ b ≥ 0}
x = a
y = b
z = 1
while (y != 0):
    while (y % 2 == 0):
        x = x * x
        y = y / 2
    y = y - 1
    z = z * x
{z = a^b}
```

Your job is to convincingly argue that this code works as intended (terminates with the correct answer.) If you're not sure how/why it works, simulating the code by hand (on various inputs) is an excellent way to gather evidence and hypotheses for useful loop invariants.

A. [10 pionts] Specify a suitable invariant I_1 for the outer while loop.

B. [10 points] Specify a suitable invariant I_2 for the inner while.

C. [10 points] Briefly explain (using words and complete sentences)

   a. Why is your invariant I_1 is true right before the outer loop begins;

   b. Why is your invariant I_2 is true right before the inner loop begins;

   c. Why does the code in the inner loop preserves the invariant I_2 (that is, why I_2 is still true at the end of the inner loop body);

   d. Why does the invariant I_2 after the inner loop (plus the fact that the inner loop terminated) prove that I_1 still holds (i.e., that the outer loop preserves the invariant I_1);

   e. Why does the invariant I_1 after the outer loop (plus the fact that the outer loop terminated) prove the desired postcondition.

D. [5 points] Give variants for both loops. (No justification is required.)

■

4. [10 points] Indy and Invy have been learning about sorting algorithms in CS 140, and have been asked to prove that a particular sorting algorithm actually works. The pseudocode given to them is:

```
// Precondition: We have an array a[0...(N-1)]
for i in 0...(N-1):
        ...code for iteration i...
// Postcondition: The array a[0...(N-1)] is sorted
```

Let $P(m)$ mean that the first m elements of array a (namely $a[0 \ldots (m-1)]$) are correctly sorted.
Indy decides to do a correctness proof by induction, showing that after k iterations of the loop (where $0 \leq k \leq N$), $P(k)$ holds. As usual for induction on numbers, he has to do two subproofs:

- the base case: $P(0)$ is true.
- the inductive step: for any k in the range $0 \ldots (N-1)$, if $P(k)$ is true, then $P(k+1)$ is true.

Since the loop will execute N times, he concludes that when the loop is done, $P(N)$ holds and the entire array $a[0 \ldots (N-1)]$ is sorted.
Invy decides to do a proof using loop invariants. Although we haven't discussed loop invariants for for loops, she realizes that the same sorting algorithm can be written using a while loop:

```
// Precondition: We have an array a[0...(N-1)]
i = 0
while (i != N):
        ...code for iteration i...
        i = i + 1
// Postcondition: The array a[0...(N-1)] is sorted
```

Invy then proves correctness of this sorting algorithm by showing that $P(i)$ is a suitable loop invariant (i.e., it is a loop invariant, and this invariant makes the rest of the proof easy).

A. [5 points] Carefully explain why Indy and Invy ended up doing basically the same work, even though they used two different proof methods.

B. [5 points] For lots of algorithms, as here, you can either do proof-by-induction or proof-by-loop-invariant. When two different approaches give you the same results with equal amounts of work, you might as well use the approach you're more comfortable or familiar with — most folks choose induction. But in what sort of situations (or what sort of algorithms, or what sort of code) might loop-invariants make more sense than induction?

■

5. [1 easy point] Answer this quick survey when you're done:

A. How long (in hours) did you spend working on this assignment?

B. What was the most interesting thing you learned while answering these problems? (Were sure there was *something* you learned.)

■