15-122 : Principles of Imperative Computation, Fall 2013

Written Homework 2

Due: Thursday, September 12, 2013 at 10pm

Name: ____________________

Andrew ID: ____________________

Recitation: ____________________

The written portion of this week's homework will give you some practice working with the binary representation of integers and reasoning with invariants. You are strongly advised to review the C0 language reference guide (available at http://c0.typesafety.net/) for details on integer manipulation.

Question	Points	Score
1	8	
2	7	
Total:	15	

You must do this assignment in one of two ways:

1) Write your answers *neatly* on this PDF, and then submit the stapled printout to the handin box Thursday before lecture or on Thursday afternoon outside of Tom Cortina's office (GHC 4117).

2) Use the TeX template at `https://whiteboard.ddt.cs.cmu.edu/#/15122-f13/assessments/71` and submit your solutions electronically at that address.

1. **Basics of C0**

(3) (a) Let x be an `int` in the C0 language. Express the following operations in C0 using only constants and the bitwise operators (`&`, `|`, `^`, `~`, `<<`, `>>`). Your answers should account for the fact that C0 uses 32-bit integers.

i. Set a equal to x, where the alpha and green components have both been set to 0, with the red and blue components left unchanged. (eg `0xAB12CE34` becomes `0x00120034`; see Section 1.1 of the Programming portion for more info)

Solution:

ii. Set b equal to x with its middle 16 bits flipped ($0 \implies 1$ and $1 \implies 0$) (eg `0xAB00FF12` becomes `0xABFF0012`)

Solution:

iii. Set c equal to x with its highest 8 bits set to 1 and with its lowest 8 bits set to 0. (eg `0xAB12CE34` becomes `0xFF12CE00`)

Solution:

iv. Set d equal to x with its highest and lowest 16 bits swapped (eg `0x1234ABCD` becomes `0xABCD1234`)

Solution:

(1) (b) Are the following two `bool` expressions equivalent in C0, assuming `x` and `y` are of type `int`? Explain your answer.

```
(x%y < 122) && (y != 0)          (y != 0) && (x%y < 122)
```

Solution:

(1) (c) Is the following code a valid way to check if $a+b+c$ overflows? If not, give values for a, b and c such that the check will return an incorrect result:

```
bool safe_add(int a, int b, int c)
{
  if (a > 0 && b > 0 && c > 0 && a + b + c < 0) return false;
  if (a < 0 && b < 0 && c < 0 && a + b + c > 0) return false;
  return true;
}
```

Solution:

(3) (d) For each of the following statements, determine whether the statement is true or false in C0. If it is true, explain why. If it is false, give a counterexample to illustrate why.

i. For every `int` x and y, $x < y$ is equivalent to $x - y < 0$

Solution:

ii. For every `int` x: x `>>` 1 is equivalent to $x/2$.

Solution:

iii. For every `int` x, y, and z: $(x + y) * z$ is equivalent to $z * y + x * z$.

Solution:

2. **Reasoning with Invariants**

The Pell sequence is shown below:

```
0, 1, 2, 5, 12, 29, 70, 169, 408, 985, ...
```

Each integer i_n in the sequence is the sum of $2i_{n-1}$ and i_{n-2}. Consider the following implementation for `fastpell` that returns the n^{th} Pell number (the body of the loop is not shown).

```
int PELL(int n)
//@requires n >= 1;
{
  if (n <= 1) return 0;
  else if (n == 2) return 1;
  else return 2 * PELL(n-1) + PELL(n-2);
}

int fastpell(int n)
//@requires n >= 1;
//@ensures \result == PELL(n);
{
  if (n <= 1) return 0;
  if (n == 2) return 1;
  int i = 1;
  int j = 0;
  int k = 2;
  int x = 3;
  while (x < n)
    //@loop_invariant 3 <= x && x <= n;
    //@loop_invariant i == PELL(x-1);
    //@loop_invariant j == PELL(x-2);
    //@loop_invariant k == 2*i+j;
    {
      // LOOP BODY NOT SHOWN
    }
  return k;
}
```

In this problem, we will reason about the correctness of the `fastpell` function when the argument `n` is greater than or equal to `3`, and we will complete the implementation based on this reasoning.

To completely reason about the correctness of `fastpell`, also need to point out that `fastpell(1) == PELL(1)` and that `fastpell(2) == PELL(2)`. This is straightforward, because no loops are involved.

(2) (a) Show that each loop invariant is true just before the loop condition is tested for the first time, using the precondition and any initialization before the loop condition.

Solution:

- `3 <= x && x <= n` – We know `x` is `3` by line $\boxed{}$, so `3 <= x` is true because `3 <= 3`.

  Because `x` is `3`, to show `x <= n` we just need to show `3 <= n` before the loop condition is tested for the first time. We know `n` is greater than 1 by line $\boxed{}$ and we know `n` is not 2 by line $\boxed{}$, so it follows that `n` is greater than or equal to 3.

- `i == PELL(x-1)` – Because `x` is `3` before the loop condition is tested for the first time, `PELL(x-1)` is $\boxed{}$ and therefore this loop invariant is initially justified by line $\boxed{}$.

- `j == PELL(x-2)` – Because `x` is `3` before the loop condition is tested for the first time, `PELL(x-2)` is $\boxed{}$ and therefore this loop invariant is initially justified by line $\boxed{}$.

- `k == 2*i+j` – Justified by lines $\boxed{}$.

(2) (b) Show that the loop invariants and the negated loop guard at termination imply the postcondition.

Solution:

We know `x <= n` by line ☐, and we know `x >= n` by line ☐, so this implies that `x` equals `n`.

The result value is the value of `k` after the loop, so to show that that the postcondition holds when `n >= 3`, it suffices to show that, after the loop, `k` equals `PELL(x)`.

- `k` = `2*i + j` (line ☐)
- `2*i + j` = `2*PELL(x-1) + j` (line ☐)
- `2*PELL(x-1) + j` = `2*PELL(x-1) + PELL(x-2)` (line ☐)
- `2*PELL(x-1) + PELL(x-2)` = `PELL(x)` (`PELL` def., `x >= 3` by line ☐)
- `k` = `PELL(x)` (transitivity, the four preceding facts)

(2) (c) Based on the given loop invariant, write the body of the loop. *DO NOT* use the specification function `PELL()`.

Solution:

```
{
  j =

  i =

  k =

  x =
}
```

(1) (d) Explain why the function must terminate with the loop you gave in 2(c).

Solution: