

15-122 : Principles of Imperative Computation, Spring 2013**Homework 1 Theory**

Due: Tuesday, February 5, 2013, at the **beginning** of lecture

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* use this printout, include this cover sheet, and staple the whole thing together before turning it in. Either type up the assignment using `15122-theory1.tex`, or print this PDF and write your answers *neatly* by hand.

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 : $x \gg 1$ 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).

```
/* 1 */ int PELL(int n)
/* 2 */ //@requires n >= 1;
/* 3 */ {
/* 4 */     if (n <= 1) return 0;
/* 5 */     else if (n == 2) return 1;
/* 6 */     else return 2 * PELL(n-1) + PELL(n-2);
/* 7 */ }
/* 8 */
/* 9 */ int fastpell(int n)
/* 10 */ //@requires n >= 1;
/* 11 */ //@ensures \result == PELL(n);
/* 12 */ {
/* 13 */     if (n <= 1) return 0;
/* 14 */     if (n == 2) return 1;
/* 15 */     int i = 1;
/* 16 */     int j = 0;
/* 17 */     int k = 2;
/* 18 */     int x = 3;
/* 19 */     while (x < n)
/* 20 */         //@loop_invariant 3 <= x && x <= n;
/* 21 */         //@loop_invariant i == PELL(x-1);
/* 22 */         //@loop_invariant j == PELL(x-2);
/* 23 */         //@loop_invariant k == 2*i+j;
/* 24 */         {
/* 25 */             // LOOP BODY NOT SHOWN
/* 26 */         }
/* 27 */     return k;
/* 28 */ }
```

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:

- $3 \leq x \ \&\& \ x \leq n$ – We know x is 3 by line , so $3 \leq x$ is true because $3 \leq 3$.

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

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

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

- $k == 2*i+j$ – Justified by lines .

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

Solution:

We know $x \leq n$ by line , and we know $x \geq n$ by line , so this implies that x equals n .

The result value is the value of k after the loop, so to show that the postcondition holds when $n \geq 3$, it suffices to show that, after the loop, k equals $\text{PELL}(x)$.

- $k = 2*i + j$ (line)
- $2*i + j = 2*\text{PELL}(x-1) + j$ (line)
- $2*\text{PELL}(x-1) + j = 2*\text{PELL}(x-1) + \text{PELL}(x-2)$ (line)
- $2*\text{PELL}(x-1) + \text{PELL}(x-2) = \text{PELL}(x)$ (PELL def., $x \geq 3$ by line)
- $k = \text{PELL}(x)$ (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 $\text{PELL}()$.

Solution:

```
{
  j =
  i =
  k =
  x =
}
```

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

Solution: