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 should type up your solutions or write them *neatly* by hand, and you should submit your work in class on the due date just before lecture begins.

Be sure to staple your homework before you submit it and make sure your name and section letter is clearly shown.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
1. Basics of C₀

(a) Let \( x \) be an int in the C₀ language. Express the following operations in C₀ using only constants and the bitwise operators (\&, |, ^, ~, <<, >>). [1 pt each]

i. Set \( a \) equal to \( x \) multiplied by 32.

 Solution:

ii. Set \( b \) equal to \( x \mod 8 \), assuming that \( x \) is positive.

 Solution:

iii. Set \( c \) equal to \( x \) with its highest 13 bits all set to 1.

 Solution:

iv. Set \( d \) equal to the intensity of the blue component of \( x \), assuming \( x \) stores the packed representation of an ARGB color (see Section 1.1 of the Programming Part). The intensity should be a value between 0 and 255, inclusive.

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

\[(x/y < 122) \&\& (y \neq 0)\]

\[(y \neq 0) \&\& (x/y < 122)\]

**Solution:**

(4)  (c) 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`: \(x > x - 1\).

**Solution:**

ii. For every `int x`: \(x \gg 1\) is equivalent to \(x/2\).

**Solution:**

iii. For every `int x`: \(x/10 * 10 + x \mod 10\) is equivalent to \(x\).

**Solution:**

iv. For every `bool a` and `bool b`: \(!a \mid\mid b\) is equivalent to \(! (a \&\& !b)\).

**Solution:**
2. **Reasoning with Invariants**

The Pell sequence is shown below:

\[0, 1, 2, 5, 12, 29, 70, 169, 408, 985, \ldots\]

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).

```c
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 && i == pell(x-1);
        //@loop_invariant j == pell(x-2) && k == 2*i+j;
        {
        // LOOP BODY NOT SHOWN
        }
    return k;
}
```

(a) Using the precondition and loop invariant, reason that the `fastpell` function must return the correct answer, even if you don’t know what is in the body of the loop. You may assume the loop invariant is correct.

**Solution:**
(2) (b) Based on the given loop invariant, write the body of the loop.

Solution:

(1) (c) Overflow occurs when the result of an integer operation cannot be represented by \texttt{int}. For instance, \(2^{30} + 2^{30}\) is \(2^{31}\), but \(2^{31}\) doesn’t have a representation in \texttt{int}. Change the body of the loop to print ”Overflow” if any of the operations overflow. State your new loop body:

Solution:

(1) (d) What is the largest Pell number representable in C0’s type \texttt{int}?

Solution:
3. More on Reasoning with Invariants

A C₀ programmer was writing a function to add up the first n positive odd integers 1, 3, …, 2n – 1 and, after testing, noticed that the first n positive odd integers always seemed to add up to \( n^2 \). To verify this, the programmer added annotations to the function as shown below:

```c
int sum_first_odd(int n)
//@requires n > 0;
//@ensures \result == n * n;
{
    int sum = 0;
    int i = n;
    while (i > 0)
        //@loop_invariant 0 <= i && i <= n;
        //@loop_invariant i*i + sum == n*n;
        {
            sum = sum + 2*i-1;
            i = i - 1;
        }
    return sum;
}
```

Prove that the postcondition (ensures) holds for the function using the given precondition (requires) and the loop invariants:

(1) (a) Give a brief argument explaining why the loop must terminate.

Solution:

(1) (b) Show that each loop invariant is true immediately before the loop condition is tested for the first time.

Solution:
(c) Show that if each loop invariant is true at the start of a loop iteration, then the loop invariants are also all true at the end of that iteration.

Solution:

(d) Show that if the loop terminates, the postcondition must hold.

Solution: