Homework 4

Due: February 4th, 11:59pm

Before you start: This homework requires some formatting. Please bold or highlight your assertions. When proving implications, follow proper proof format of proof by calculation, as shown in lecture and section.

Task 1 – Reason Your Chicken [16 pts]

In this problem, you will practice proving correctness of straight-line code using forward and backward reasoning. Remember to **bold or highlight your assertions**.

a) Use **forward** reasoning to fill in the assertions. Then, show that the code is correct by proving by **cases** that P implies Q.

$\{\{\, x \geqslant 0 \,\}\}$
```
if (x >= 10) {
  {{ ________________________________ }}
  y = 4 * x - 12;
  {{ ________________________________ }}
} else {
  {{ ________________________________ }}
  y = 18 - x;
  {{ ________________________________ }}
}
```
$\{\{\, P :$ ______________________ or ______________________ $\}\}$

$\{\{\, Q :\ y \geqslant 4 \,\}\}$

b) Use **forward** reasoning to fill in P_1 and P_2. Use **backward** reasoning to fill in Q_1 and Q_2 and other assertions.

Fill in each blank by applying the rules *exactly* as taught in lecture. You may simplify the resulting assertion, but do not weaken it. If you simplify an assertion, use "$\leftrightarrow$" to separate the simplified assertion from the original assertion, as we did in section.)

Then, show that the code is correct by proving that P_1 implies Q_1 and P_2 implies Q_2.

$\{\{\, P : \; x \geqslant 0 \,\}\}$

```
if (x >= 10) {
```

$\{\{\, P_1 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \,\}\}$

$\{\{\, Q_1 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \,\}\}$

```
  y = 4 * x - 12;
```

$\{\{\, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \,\}\}$

```
} else {
```

$\{\{\, P_2 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \,\}\}$

$\{\{\, Q_2 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \,\}\}$

```
  y = 18 - x;
```

$\{\{\, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \,\}\}$

```
}
```

$\{\{\, y \geqslant 4 \,\}\}$

Task 2 – The Choice is Hoare's [12 pts]

When doing forward or backward reasoning, fill in each blank, making sure to **bold or highlight your assertions**. You may simplify the resulting assertion, but do not weaken it. Separate any simplified statement from the original with "$\leftrightarrow$".

In the code below, we use the "::" operator as a shorthand for `cons`. Recall that the function sum : (List) $\to \mathbb{Z}$ is defined by

$$\text{sum}(\text{nil}) := 0$$
$$\text{sum}(x :: L) := x + \text{sum}(L)$$

a) Use **forward** reasoning to fill in the missing assertions in the following code:

Avoid subscripts in your assertions. We have designed this problem so that subscripts are not necessary.

$\{\{\, x > 0 \text{ and } L = 7 :: \text{nil} \,\}\}$
```
x = x + 12;
```
$\{\{$ ____ $\}\}$
```
R = cons(x, L);
```
$\{\{$ ____ $\}\}$
```
y = x - 12;
```
$\{\{\, P :$ ______________________________________ $\}\}$
$\{\{\, Q : \text{ sum}(R) > 19 \,\}\}$

b) Show that the code is correct by proving by calculation that P implies Q.

Continued on the next page...

c) Use **backward** reasoning to fill in the missing assertions in the following code:

$\{\{\, P: \ x > 0 \text{ and } L = 6 :: \mathsf{nil} \,\}\}$
$\{\{\, Q: \ \underline{\hspace{8cm}} \,\}\}$
```
x = x + 7;
```
$\{\{\, \underline{\hspace{9cm}} \,\}\}$
```
L = cons(x, L);
```
$\{\{\, \underline{\hspace{9cm}} \,\}\}$
```
L = cons(8, L);
```
$\{\{\, \mathsf{sum}(L) > 21 \,\}\}$

d) Show that the code is correct by proving by calculation that P implies Q.

Task 3 – Loops, I Did It Again [13 pts]

Suppose we define the set of binary digits as

$$\textbf{type } \mathsf{Digit} := 0 \mid 1$$

Then, we can represent a number written in binary as the following list type:

$$\textbf{type } \mathsf{Digits} := \mathsf{bitEnd}(\mathsf{Digit}) \mid \mathsf{bitCons}(\mathsf{Digit}, \mathsf{Digits})$$

bitEnd represents the last bit, and bitCons functions the same as our List cons for the binary number.

The following function, toInt : Digits $\to \mathbb{Z}$ translates a sequence of binary digits into the corresponding integer:

$$\begin{aligned} \mathsf{toInt}(\mathsf{bitEnd}(d)) &:= d \\ \mathsf{toInt}(\mathsf{bitCons}(d, R)) &:= d + 2\,\mathsf{toInt}(R) \end{aligned}$$

For example, we can see that

$$\begin{aligned} &\mathsf{toInt}(\mathsf{bitCons}(1, \mathsf{bitCons}(0, \mathsf{bitCons}(1, \mathsf{bitEnd}(0))))) \\ &= 1 + 2\,\mathsf{toInt}(\mathsf{bitCons}(0, \mathsf{bitCons}(1, \mathsf{bitEnd}(0)))) \\ &= 1 + 2(0 + 2\,\mathsf{toInt}(\mathsf{bitCons}(1, \mathsf{bitEnd}(0)))) \\ &= 1 + 2(0 + 2(1 + 2\,\mathsf{toInt}(\mathsf{bitEnd}(0)))) \\ &= 1 + 2(0 + 2(1 + 2 \cdot 0)) \\ &= 1 + 2(0 + 2 \cdot 1) \\ &= 1 + 2 \cdot 2 \\ &= 5 \end{aligned}$$

Note that the binary representation of 5 is usually written 0101, not 1010, so this definition expects the digits to be stored in the list in reversed order, with the *least* significant digit at the front. Such a representation is called "little endian" (while the ordinary representation is "big endian").

We can represent the Digits data type as this Java class:

```
public class Digits {
    public int digit;
    public Digits next;
}
```

where `next` is `null` if the node is a "bitEnd" and non-`null` if it is a "bitCons".

The following code takes the digits in the variable `L` and claims to compute $\text{toInt}(L)$ in the variable `v`. The variable d refers to the digit stored in `L.digit` and R refers to `L.next` in math notation.

$\{\{\ L = L_0\ \}\}$

```
int b = 1;
int v = 0;
```

$\{\{\ \text{Inv: toInt}(L_0) = v + b \cdot \text{toInt}(L)\ \}\}$

```
while (L.next != null) {
```

$\{\{\ P_1:\ L = \text{bitCons}(d, R) \text{ and } \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\ \}\}$

```
    v = v + b * L.digit;
```

$\{\{\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\ \}\}$

```
    b = 2 * b;
```

$\{\{\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\ \}\}$

```
    L = L.next;
```

$\{\{\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\ \}\}$

$\{\{\ \text{Inv: toInt}(L_0) = v + b \cdot \text{toInt}(L)\ \}\}$

```
}
```

$\{\{\ P_2:\ L = \text{bitEnd}(d) \text{ and } \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\ \}\}$

```
v = v + b * L.digit;
```

$\{\{\ v = \text{toInt}(L_0)\ \}\}$

Inside the body of the loop, since `L.next != null`, we can write $L = \text{bitCons}(d, R)$ for some d and R. Likewise, after the loop, the assertion will start with $L = \text{bitEnd}(d)$ for some d, since `L.next == null`.

a) Prove that the invariant is true right before we enter the loop.

b) What are the known facts at P_2, when we exit the loop?

c) Prove that the postcondition holds. Use either forward or backward reasoning (your choice) on the line of code after the loop, and indicate which reasoning you chose. Then, check that the resulting implication holds. You **must** state your choice of reasoning and include your assertions.

Remember that `L.digit` is called "d" in the math.

d) What are the known facts at P_1, when we enter the loop body?

e) Prove that the invariant is preserved by the body of the loop. Use either forward or backward reasoning (but not a mix of the two) to reduce the body to an implication. Then, prove the implication. Again, you **must** state your choice of reasoning and include your assertions.

Hint for forward reasoning: The line `L = L.next` turns L into $\text{bitCons}(d, L)$.

Hint for backward reasoning: Remember that `L.next` is called "R" in math notation.

Task 4 – Sum-thing Borrowed, Sum-thing Blue [12 pts]

The function mult : ($\mathbb{Z}$, List) → List multiplies each element in a list of integers by a given multiplier:

$$\begin{aligned} \text{mult}(x, \text{nil}) &:= \text{nil} \\ \text{mult}(x, m :: L) &:= x \cdot m :: \text{mult}(x, L) \end{aligned}$$

Code snippets A and B below both claim to compute the sum of a list of integers whose elements were multiplied by a factor of 4:

A.

$\{\{\, L = L_0 \,\}\}$

```
int z = 0;
int x = 4;
```

$\{\{\, \text{Inv: } \text{sum}(\text{mult}(x, L_0)) = z + \text{sum}(\text{mult}(x, L)) \,\}\}$

```
while (L != null) {
```

$\{\{\, P_0 : \underline{\hspace{8cm}} \,\}\}$

$\{\{\, Q_2 : \underline{\hspace{8cm}} \,\}\}$

```
    z = z + L.hd * x;
```

$\{\{\, Q_1 : \underline{\hspace{8cm}} \,\}\}$

```
    L = L.tl;
```

$\{\{\, Q_0 : \underline{\hspace{8cm}} \,\}\}$

```
}
```

$\{\{\, z = \text{sum}(\text{mult}(x, L_0)) \,\}\}$

B.

$\{\{\, L = L_0 \,\}\}$

```
int d = 0;
int v = 4;
```

$\{\{\, \text{Inv: } \text{sum}(L_0) = d + \text{sum}(L) \,\}\}$

```
while (L != null) {
```

$\{\{\, P_0 : \underline{\hspace{8cm}} \,\}\}$

$\{\{\, Q_2 : \underline{\hspace{8cm}} \,\}\}$

```
    d = d + L.hd;
```

$\{\{\, Q_1 : \underline{\hspace{8cm}} \,\}\}$

```
    L = L.tl;
```

$\{\{\, Q_0 : \underline{\hspace{8cm}} \,\}\}$

```
}
d = d * v
```

$\{\{\, d = v \cdot \text{sum}(L_0) \,\}\}$

a) Fill in the remaining assertions for A. Fill in P_0 and Q_0 using the Floyd logic rule for while loops. Use **backward reasoning** to derive Q_1 and Q_2. Remember to **bold or highlight your assertions**!

b) Does P_0 imply Q_2 in snippet A? Informally explain why or why not. (No proof required.)

c) Fill in the remaining assertions for B, in the same manner as part (a).

d) Does P_0 imply Q_2 in snippet B? Informally explain why or why not. (No proof required.)

e) Is the value in z at the end of snippet A equal to the value in d at the end of snippet B? Informally explain your answer in 1 or 2 sentences. (No proof required.)

Task 5 – Optional: Chicken Noodle Loop [0 pts]

In this problem, we will prove the correctness one version of the code for `log3` produced by AI in Homework 1. Here is the code produced by Cursor's chat agent:

$\{\{\, n \geqslant 1 \,\}\}$

```
int k = 0;
int m = 1;
```

$\{\{$ Inv: $m = 3^k$ and $m < 3n \,\}\}$

```
while (m < n) {
   m = 3 * m;
   k = k + 1;
}
```

$\{\{\, 3^{k-1} < n$ and $n \leqslant 3^k \,\}\}$

The postcondition comes from the specification, specifically the `@return` tag, which said that the value of k returned should satisfy these two inequalities.

The loop invariant was not given in the specification, nor was it provided by the AI. I have filled it in for you to make it possible to complete the proof. (More on this later...)

a) Prove that the invariant is true when we get to the top of the loop the first time.

b) Prove that, when we exit the loop, the postcondition holds.

c) Prove that the invariant is preserved by the body of the loop. Use either forward or backward reasoning (your choice) to reduce the body to an implication and then check that it holds.

d) Would you have guessed that was the loop invariant just by looking at the code?

If the author of the code (AI, in this case) had proven the code correct, do you think they should have included the loop invariant in the comments or is it fine for them to leave it out and let you figure it out for yourself?