Quiz Section 6: Imperative Programming

Task 1 – It's Forward Against Mine

In this problem, we will practice using forward reasoning to check the correctness of assignments. Assume that all variables are `bigints`. Do not use subscripts for this problem (they are not necessary), instead write assertions in terms of the current values of variables.

(a) Use forward reasoning to fill in the missing assertions (strongest postconditions) in the following code. Then prove that the stated postcondition holds.

$\{\{\, x \geqslant 4 \,\}\}$
```
y = x - 2n;
```
$\{\{$ _____ $\}\}$
```
z = 2n * y;
```
$\{\{$ _____ $\}\}$
```
z = z - 2n;
```
$\{\{$ _____ $\}\}$
$\{\{\, z \geqslant 0 \,\}\}$

(b) Use forward reasoning to fill in the missing assertions (strongest postconditions) in the following code. Then prove that the stated postcondition holds. (Reminder that with `bigint`, division is truncating division.)

$\{\{\, x \leqslant 4 \,\}\}$
```
y = x + 4n;
```
$\{\{$ _____ $\}\}$
```
x = x / 2n;
```
$\{\{$ _____ $\}\}$
```
y = y + 2 * x;
```
$\{\{$ _____ $\}\}$
$\{\{\, y < 14 \,\}\}$

Task 2 – Not For a Back of Trying

In this problem, we will practice using backward reasoning to check the correctness of assignments. Assume that all variables are `bigints`.

(a) Use backward reasoning to fill in the missing assertions (weakest preconditions) in the following code. Then prove that the stated precondition implies what is required for the code to be correct.

Feel free to simplify the intermediate assertions (i.e., rewrite them in an equivalent, but simpler, way). However, the assertions you write must be equivalent to still be weakest preconditions.

$\{\{\, x < w + 1 \,\}\}$
$\{\{$ _____ $\}\}$
```
y = 3n * w;
```
$\{\{$ _____ $\}\}$
```
x = x * 3n;
```
$\{\{$ _____ $\}\}$
```
z = x - 9n;
```
$\{\{\, z < y \,\}\}$

(b) Use backward reasoning to fill in the missing assertions (weakest preconditions) in the following code. Then prove that the stated precondition implies what is required for the code to be correct.

Feel free to simplify the intermediate assertions (i.e., rewrite them in an equivalent, but simpler, way). However, the assertions you write must be equivalent to still be weakest preconditions.

$\{\{\, x > 1 \,\}\}$
$\{\{$ _____ $\}\}$
```
y = x - 4n;
```
$\{\{$ _____ $\}\}$
```
z = 3n * y;
```
$\{\{$ _____ $\}\}$
```
z = z + 6n;
```
$\{\{\, z \geqslant y \,\}\}$

Task 3 – Nothing to Be If-ed At

In this problem, we will practice using forward reasoning to check correctness of `if` statements. Assume that all variables are `bigints`.

(a) Use forward reasoning to fill in the assertions. Then, combine the branches to assert the invariant we know at the end of the conditional and complete an argument by cases that this invariant implies $\{\{y \geqslant 2\}\}$.

Assume that `x` and `y` are both integers.

$\{\{ x \geqslant 0 \}\}$
```
if (x >= 6n) {
```
$\{\{$ _____ $\}\}$
```
  y = 2n * x - 10n;
```
$\{\{$ _____ $\}\}$
```
} else {
```
$\{\{$ _____ $\}\}$
```
  y = 20n - 3n * x;
```
$\{\{$ _____ $\}\}$
```
}
```
$\{\{$ _____ or _____ $\}\}$
$\{\{ y \geqslant 2 \}\}$

(b) Use forward reasoning to fill in the assertions. Then, combine the branches to assert the invariant we know at the end of the conditional and complete an argument by cases that this invariant implies $\{\{s \geqslant 1\}\}$.

Assume that `s` and `t` are both integers.

$\{\{\ s \neq t$ and $t > 0\ \}\}$
```
if (s >= t) {
```
$\{\{$ _____ $\}\}$
```
  s = s / t;
```
$\{\{$ _____ $\}\}$
```
} else {
```
$\{\{$ _____ $\}\}$
```
  s = t - s;
```
$\{\{$ _____ $\}\}$
```
}
```
$\{\{$ _____ or _____ $\}\}$

$\{\{\ s \geqslant 1\ \}\}$

Task 4 – The Only Game in Down

The function "countdown" takes an integer argument "n" and returns a list containing the numbers $n, \ldots, 1$. It can be defined recursively as follows:

$$\text{countdown} : \mathbb{N} \to \text{List}$$

$$\begin{aligned} \text{countdown}(0) &:= \text{nil} \\ \text{countdown}(n+1) &:= (n+1) :: \text{countdown}(n) \end{aligned}$$

This function is defined recursively on a natural number so it fits the natural number template from lecture. In this problem, we will prove the following code correctly calculates countdown(n). The invariant for the loop is already provided.

```
let i: bigint = 0;
let L: List = nil;
{{ Inv: L = countdown(i) }}
while (i !== n) {
   i = i+1;
   L = cons(i, L);
}
{{ L = countdown(n) }}
```

(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. To do this, use backward reasoning to reason until the statement "`i = i + 1;`". Then complete the correctness check by verifying that the invariant with the loop condition implies the assertion you produced with backward reasoning.

Task 5 – Chicken Noodle Loop

The function sum-abs calculates the sum of the absolute values of the numbers in a list. We can give it a formal definition as follows:

$$\text{sum-abs} : \text{List} \to \mathbb{Z}$$

$$\begin{aligned}
\text{sum-abs}(\text{nil}) &:= 0 \\
\text{sum-abs}(x :: L) &:= -x + \text{sum-abs}(L) \quad \text{if } x < 0 \\
\text{sum-abs}(x :: L) &:= x + \text{sum-abs}(L) \quad \text{if } x \geqslant 0
\end{aligned}$$

In this problem, we will prove that the following code correctly calculates sum-abs(L). The invariant for the loop is already provided. It references L_0, which is the initial value of L when the function starts.

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== ''nil'') {
   if (L.hd < 0n) {
      s = s + -L.hd;
   } else {
      s = s + L.hd;
   }
   L = L.tl;
}
{{ s = sum-abs(L_0) }}
```

(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. To do this, use backward reasoning to reason through the last assignment statement "`L = L.tl;`". Then, use forward reasoning for each branch of the "`if`" statement (as in Problem 3). Finally, complete the correctness check by verifying that each of the assertions you produced with forward reasoning implies the assertion produced by backward reasoning immediately above the last assignment statement.

We have previously used the fact that, when $L \neq \text{nil}$, we know that $L = \text{cons}(x, R)$ for some $x : \mathbb{Z}$ and $R : \text{List}$. However, in the code, we know exactly what x and R are, namely, $x = L.\text{hd}$ and $R = L.\text{tl}$. Hence, when $L \neq \text{nil}$, we actually have $L = \text{cons}(L.\text{hd}, L.\text{tl})$. Feel free to use that in your proof.