CSE 331: Software Design & Implementation

Section 5

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 represent integers.

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

$\{\{\, x \geq 3 \,\}\}$
```
y = x - 2;
```
$\{\{$ _____ $\}\}$
```
z = 2 * y;
```
$\{\{$ _____ $\}\}$
```
z = z - 2;
```
$\{\{$ _____ $\}\}$
$\{\{\, z \geq 0 \,\}\}$

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

$\{\{\, x < 3 \,\}\}$
```
y = x + 4;
```
$\{\{$ _____ $\}\}$
```
x = 2 * x;
```
$\{\{$ _____ $\}\}$
```
y = y + x;
```
$\{\{$ _____ $\}\}$
$\{\{\, y < 14 \,\}\}$

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 represent integers.

(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 = 3 * w;
```
$\{\{$ _____ $\}\}$
```
x = x * 3;
```
$\{\{$ _____ $\}\}$
```
z = x - 6;
```
$\{\{ z < y \}\}$

(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 - 4;
```
$\{\{$ _____ $\}\}$
```
z = 3 * y;
```
$\{\{$ _____ $\}\}$
```
z = z + 6;
```
$\{\{ z \geq y \}\}$

3. Nothing to Be If-ed At

In this problem, we will practice using forward and backward reasoning to check correctness of `if` statements. Assume that all variables represent integers.

(a) Use forward reasoning to fill in P_1 and P_2 with the strongest postconditions at these two lines. Then, use backward reasoning to fill in Q_1 and Q_2 with the weakest preconditions at those lines. Finally, complete the correctness proof by showing that P_1 implies Q_1 and P_2 implies Q_2.

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

$\{\{\, s \neq t \text{ and } t \neq 0 \,\}\}$
```
if (s >= t) {
```
$\{\{\, P_1 : \_\_\_\_\_ \,\}\}$
$\{\{\, Q_1 : \_\_\_\_\_ \,\}\}$
```
  s = s / t;
```
$\{\{\, \_\_\_\_\_ \,\}\}$
```
} else {
```
$\{\{\, P_2 : \_\_\_\_\_ \,\}\}$
$\{\{\, Q_2 : \_\_\_\_\_ \,\}\}$
```
  s = t - s;
```
$\{\{\, \_\_\_\_\_ \,\}\}$
```
}
```
$\{\{\, s \geq 1 \,\}\}$

(b) Use forward reasoning to fill in P_1 and P_2 with the strongest postconditions at these two lines. Then, use backward reasoning to fill in Q_1 and Q_2 with the weakest preconditions at those lines. Finally, complete the correctness proof by showing that P_1 implies Q_1 and P_2 implies Q_2.

$\{\{\, x \geq 0 \,\}\}$
```
if (x >= 6) {
```
$\{\{\, P_1 : \_\_\_\_\_ \,\}\}$
$\{\{\, Q_1 : \_\_\_\_\_ \,\}\}$
```
  y = 2*x - 10;
```
$\{\{\, \_\_\_\_\_ \,\}\}$
```
} else {
```
$\{\{\, P_2 : \_\_\_\_\_ \,\}\}$
$\{\{\, Q_2 : \_\_\_\_\_ \,\}\}$
```
  y = 20 - 3*x;
```
$\{\{\, \_\_\_\_\_ \,\}\}$
```
}
```
$\{\{\, y > 1 \,\}\}$

4. 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:

$$
\begin{array}{lcll}
\textbf{func}\ \text{sum-abs}(\text{nil}) & := & 0 & \\
\quad \text{sum-abs}(\text{cons}(x, L)) & := & -x + \text{sum-abs}(L) & \text{if } x < 0 \quad \text{for any } x : \mathbb{Z} \text{ and } L : \text{List} \\
\quad \text{sum-abs}(\text{cons}(x, L)) & := & x + \text{sum-abs}(L) & \text{if } x \geq 0 \quad \text{for any } x : \mathbb{Z} \text{ and } L : \text{List}
\end{array}
$$

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: number = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L !== nil) {
   if (L.hd < 0) {
      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. You can do this by any combination of forward and backward reasoning.

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.

5. 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:

$$\begin{array}{lcll} \textbf{func}\ \mathsf{countdown}(0) & := & \mathsf{nil} & \\ \quad \mathsf{countdown}(n+1) & := & \mathsf{cons}(n+1, \mathsf{countdown}(n)) & \text{for any } n : \mathbb{N} \end{array}$$

This function is defined recursively on a natural number so it fits the "bottom-up" template from lecture.

(a) Using the template described in lecture, give the invariant for a loop implementation of this function.

Assume that the variable counting up to n is called "`j`" and the partial-result is stored in a variable called "`L`" (rather than "s").

(b) How do we initialize `j` and `L` so that the invariant is true initially?

(c) When do we exit the loop? What should the condition of the `while` be?

(d) What code do we write in the body of loop so that the invariant remains true and j is increased by one?

Be careful! It's easy to make a mistake here.

6. Take It From the Swap

Recall the function "swap", which swaps adjacent elements in a list:

$$
\begin{array}{lcll}
\textbf{func}\ \mathsf{swap}(\mathsf{nil}) & := & \mathsf{nil} & \\
\quad \mathsf{swap}(\mathsf{cons}(a, \mathsf{nil})) & := & \mathsf{cons}(a, \mathsf{nil}) & \text{for any } a : \mathbb{Z} \\
\quad \mathsf{swap}(\mathsf{cons}(a, \mathsf{cons}(b, L))) & := & \mathsf{cons}(b, \mathsf{cons}(a, \mathsf{swap}(L))) & \text{for any } a, b : \mathbb{Z} \text{ and } L : \mathsf{List}
\end{array}
$$

This function is defined recursively on a list argument so it fits the "top-down" template from lecture.

For simplicity, we will require the list to always have an even number of elements. Thus, the middle case in the above definition will not occur.

(a) Using the template described in lecture, give the invariant for a loop implementation of this function.

As in the template, we will have a variable "R" that stores the partially-completed answer (reversed).

(b) How do we initialize R so that the invariant is true initially?

(c) When do we exit the loop? What should the condition of the `while` be?

(d) The template tells us to move down the list by setting `L = L.tl`. However, swap makes its recursive call on a list that is shorter by two elements, so our loop body should try to move forward by two elements.

What code do we write so that the list gets shorter by two elements and the invariant remains true?

Be careful! Remember that R stores the partial answer in *reverse* order!