CSE 331: Software Design & Implementation Spring 2025

Quiz Section 6: Imperative Programming – Solutions

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 `bigint`s. Do not use subscripts for this problem unless otherwise specified, 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;
$\{\{$ ________________________________ $\}\}$
$\{\{\ z \geqslant 0\ \}\}$
```

```
$\{\{\ x \geqslant 4\ \}\}$
 y = x - 2n;
$\{\{\ x \geqslant 4$ and $y = x - 2\ \}\}$
 z = 2n * y;
$\{\{\ x \geqslant 4$ and $y = x - 2$ and $z = 2y\ \}\}$
 z = z - 2n;
$\{\{\ x \geqslant 4$ and $y = x - 2$ and $z + 2 = 2y\ \}\}$
```

We can see that this last assertion implies the stated postcondition $z \geqslant 0$ as follows:

$$\begin{aligned}
z &= 2y - 2 && \text{since } z + 2 = 2y \\
&= 2(x - 2) - 2 && \text{since } y = x - 2 \\
&= 2x - 6 \\
&\geqslant 2 \cdot 4 - 6 && \text{since } x \geqslant 4 \\
&= 2
\end{aligned}$$

Thus, we know $z \geqslant 2$ which implies $z \geqslant 0$.

(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.) You may use subscripts for this part.

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

$\{\{\, x \leqslant 4 \,\}\}$
`y = x + 4n;`
$\{\{\, x \leqslant 4$ and $y = x + 4 \,\}\}$
`x = x / 2n;`
$\{\{\, x_0 \leqslant 4$ and $x = x_0/2$ and $y = x_0 + 4 \,\}\}$
`y = y + 2n * x;`
$\{\{\, x_0 \leqslant 4$ and $x = x_0/2$ and $y - 2x = x_0 + 4 \,\}\}$
$\{\{\, y < 14 \,\}\}$

We can see that this last assertion implies the stated postcondition $y < 14$ as follows:

$$\begin{aligned} y &= 2x + x_0 + 4 && \text{since } y - 2x = x_0 + 4 \\ &= 2(x_0/2) + x_0 + 4 && \text{since } x = x_0/2 \\ &\leqslant 2(4/2) + 4 + 4 && \text{since } x_0 \leqslant 4 \\ &= 12 \\ &< 14 \end{aligned}$$

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`. Do not use subscripts for this problem unless otherwise specified, instead write assertions in terms of the current values of variables.

(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.

$\{\{\ x < w + 1\ \}\}$

$\{\{\ \underline{\hspace{10cm}}\ \}\}$

`y = 3n * w;`

$\{\{\ \underline{\hspace{10cm}}\ \}\}$

`x = x * 3n;`

$\{\{\ \underline{\hspace{10cm}}\ \}\}$

`z = x - 9n;`

$\{\{\ z < y\ \}\}$

$\{\{\ x < w + 1\ \}\}$

$\{\{\ 3x - 9 < 3w\ \}\} \quad \rightarrow \quad \{\{\ 3x < 3w + 9\ \}\} \quad \rightarrow \quad \{\{\ x < w + 3\ \}\}$

`y = 3n * w;`

$\{\{\ 3x - 9 < y\ \}\}$

`x = x * 3n;`

$\{\{\ x - 9 < y\ \}\}$

`z = x - 9n;`

$\{\{\ z < y\ \}\}$

We can see that $x < w + 1$ implies the condition we need since

$$\begin{aligned} x &< w + 1 \\ &< (w + 1) + 2 && \text{adding 2 to } w + 1 \text{ is bigger than } w + 1 \\ &= w + 3 \end{aligned}$$

(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;`

$\{\{\, z \geqslant y \,\}\}$

$\{\{\, x > 1 \,\}\}$

$\{\{\, x - 4 \geqslant -3 \,\}\} \quad \rightarrow \quad \{\{\, x \geqslant 1 \,\}\}$

`y = x - 4n;`

$\{\{\, 3y + 6 \geqslant y \,\}\} \quad \rightarrow \quad \{\{\, 2y \geqslant -6 \,\}\} \quad \rightarrow \quad \{\{\, y \geqslant -3 \,\}\}$

`z = 3n * y;`

$\{\{\, z + 6 \geqslant y \,\}\}$

`z = z + 6n;`

$\{\{\, z \geqslant y \,\}\}$

We can see immediately from the simplifications above that $x > 1$ implies $\geqslant 1$.

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 `bigint`s. Do not use subscripts for this problem unless otherwise specified, instead write assertions in terms of the current values of variables.

(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 ______________________________ }}
{{ $y \geqslant 2$ }}
```

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

We'll prove by cases that {{ ($x \geqslant 6$ and $y = 2x - 10$) or ($x \geqslant 0$ and $x < 6$ and $y = 20 - 3x$)}} implies the post condition {{$y \geqslant 2$}}:

Assuming $(x \geqslant 6 \text{ and } y = 2x - 10)$, we have $y = 2x - 10$. We can show:

$$\begin{aligned} y &= 2x - 10 \\ &\geqslant 2 \cdot 6 - 10 \quad \text{since } x \geqslant 6 \\ &= 2 \end{aligned}$$

Assuming $(x \geqslant 0 \text{ and } x < 6 \text{ and } y = 20 - 3x)$, first note that $x < 6$ means $-x > -6$, which means that $-3x > -18$. Then, we can calculate:

$$\begin{aligned} y &= 20 - 3x \\ &> 20 - 18 \quad \text{since } -3x > -18 \\ &= 2 \end{aligned}$$

(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 $\{\{s \geq 1\}\}$. You may use subscripts for this part.

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 __________________________________ }}
{{ $s \geq 1$ }}
```

```
{{ $s \neq t$ and $t > 0$ }}
if (s > t) {
  {{ $t > 0$ and $s > t$ }}
   s = s / t;
  {{ $t > 0$ and $s_0 > t$ and $s = s_0/t$ }}
} else {
  {{ $t > 0$ and $s < t$ }}
   s = t - s;
  {{ $t > 0$ and $t - s < t$ }}
}
{{ ($t > 0$ and $s_0 > t$ and $s = s_0/t$) or ($t > 0$ and $t - s < t$) }}
{{ $s \geq 1$ }}
```

We'll prove by cases that $\{\{\, (t > 0$ and $ts > t)$ or $(t > 0$ and $t - s < t)\,\}\}$ implies the post condition $\{\{s \geq 1\}\}$:

Assuming $(t > 0$ and $s_0 > t$ and $s = s_0/t)$, we can show:

$$\begin{aligned} s &= s_0/t \\ &\geq t/t \qquad \text{since } s_0 > t \\ &= 1 \end{aligned}$$

Assuming $(t > 0$ and $t - s < t)$, we can rearrange $t - s < t$ (by adding s to both sides and then adding $-t$ to both sides) to get $0 < s$. Since s is an integer, this means $1 \leq s$, or equivalently, $s \geq 1$.

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: $L = \text{countdown}(i)$ }}
while (i !== n) {
   i = i+1;
   L = cons(i, L);
}
{{ $L = \text{countdown}(n)$ }}
```

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

At the top of the loop initially, we can see that $L = \text{nil}$ and that $i = 0$, so we have

$$\begin{aligned} L &= \text{nil} \\ &= \text{countdown}(0) && \text{def of countdown} \\ &= \text{countdown}(i) && \text{since } i = 0 \end{aligned}$$

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

After the loop we know that $i = n$ and that $L = \text{countdown}(i)$. Thus, we can see

$$\begin{aligned} L &= \text{countdown}(i) && \text{as noted above} \\ &= \text{countdown}(n) && \text{since } i = n \end{aligned}$$

(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.

We can start by filling in the assertions as follows:

$$\{\{\ L = \text{countdown}(i) \text{ and } i \neq n\ \}\}$$
$$\{\{\ (i+1) :: L = \text{countdown}(i+1)\ \}\}$$

```
i = i + 1;
```

$$\{\{\ i :: L = \text{countdown}(i)\ \}\}$$

```
L = cons(i, L);
```

$$\{\{\ L = \text{countdown}(i)\ \}\}$$

To prove that $L = \text{countdown}(i)$ implies $(i+1) :: L = \text{countdown}(i+1)$, we can calculate:

$$\begin{aligned} (i+1) :: L &= (i+1) :: \text{countdown}(i) && \text{since } L = \text{countdown}(i) \\ &= \text{countdown}(i+1) && \text{def of countdown} \end{aligned}$$

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:

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

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

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

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

At the top of the loop initially, we can see that $L = L_0$ and $s = 0$, so we have

$$\begin{aligned}
s + \text{sum-abs}(L) &= 0 + \text{sum-abs}(L) && \text{since } s = 0 \\
&= \text{sum-abs}(L_0) && \text{since } L = L_0
\end{aligned}$$

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

After the loop we know that $L = \text{nil}$ and that $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$. Thus, we can see

$$\begin{aligned}
\text{sum-abs}(L_0) &= s + \text{sum-abs}(L) && \text{as noted above} \\
&= s + \text{sum-abs}(\text{nil}) && \text{since } L = \text{nil} \\
&= s + 0 && \text{Def of sum-abs} \\
&= s
\end{aligned}$$

(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.

We can start by filling in the assertions as follows:

```
{{ $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ }}
 if (L.hd < 0n) {
   {{ $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} < 0$ }}
    s = s + -L.hd;
   {{ $s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} < 0$ }}
 } else {
   {{ $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} \geqslant 0$ }}
    s = s + L.hd;
   {{ $s - L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} \geqslant 0$ }}
 }
{{ ($s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} < 0$) }}
   {{  or ($s - L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} \geqslant 0$) }}
{{ $s + \text{sum-abs}(L.\text{tl}) = \text{sum-abs}(L_0)$ }}
 L = L.tl;
{{ $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ }}
```

We'll prove by cases that {{ ($s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} < 0$) or ($s - L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} \geqslant 0$) }} implies the post condition {{$s + \text{sum-abs}(L.\text{tl}) = \text{sum-abs}(L_0)$ }}:

Assuming ($s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} < 0$), we can calculate:

$$\begin{aligned} \text{sum-abs}(L_0) &= s + L.\text{hd} + \text{sum-abs}(L) \\ &= s + L.\text{hd} + \text{sum-abs}(L.\text{hd} :: L.\text{tl}) && \text{since } L \neq \text{nil} \\ &= s + L.\text{hd} - L.\text{hd} + \text{sum-abs}(L.\text{tl}) && \text{Def of sum-abs since } L.\text{hd} < 0 \\ &= s + \text{sum-abs}(L.\text{tl}) \end{aligned}$$

Assuming ($s - L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ and $L \neq \text{nil}$ and $L.\text{hd} \geqslant 0$), we can calculate:

$$\begin{aligned}
\textsf{sum-abs}(L_0) &= s - L.\textsf{hd} + \textsf{sum-abs}(L) \\
&= s - L.\textsf{hd} + \textsf{sum-abs}(L.\textsf{hd} :: L.\textsf{tl}) && \text{since } L \neq \textsf{nil} \\
&= s - L.\textsf{hd} + L.\textsf{hd} + \textsf{sum-abs}(L.\textsf{tl}) && \text{Def of sum-abs since } L.\textsf{hd} \geqslant 0 \\
&= s + \textsf{sum-abs}(L.\textsf{tl})
\end{aligned}$$