Quiz Section 3: Floyd Logic – Solutions

Task 1 – Found Guilty of Reason

In this problem, you will practice proving correctness of straight-line code using **forward reasoning**.

Fill in each blank by applying the rules *exactly* as taught in lecture. Then, if you want, you can simplify the resulting assertion, but do not weaken it (such as substituting/dropping facts). Separate any simplified statement from the original by "↔".

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

```
{{ y > 5 and z > 2 }}
x = 4 * y - 3;
{{ ____________________________________ }}
y = y - 5;
{{ ____________________________________ }}
z = z * y;
{{ P: _________________________________ }}
{{ Q: x < 2z + 20 }}
```

```
{{ y > 5 and z > 2 }}
x = 4 * y - 3;
{{ y > 5 and z > 2 and x = 4y - 3 }}
y = y - 5;
{{ y + 5 > 5 and z > 2 and x = 4(y + 5) - 3 }}
z = z * y;
{{ P: y + 5 > 5 and z / y > 2 and x = 4(y + 5) - 3 }}
{{ Q: x < 2z + 20 }}
```

b) Show that the code is correct by explaining how P implies Q.

We can see that Q holds since

$$\begin{aligned} x &= 4y + 17 && \text{since } x = 4(y+5) - 3 \\ &< 2z + 17 && \text{since } z > 2y \text{ because } y > 0 \\ &< 2z + 20 \end{aligned}$$

Alternate solution: We know that $x = 4(y+5) - 3 = 4y + 20 - 3 = 4y + 17$. Next, we can rewrite $z/y > 2$ into $z > 2y$ since we know that $y > 0$ (and therefore multiplying both sides by y doesn't flip the sign). We can then substitute into our original equation that $x < 2z + 17$. Finally, we know that $17 < 20$ so $x < 2z + 20$. Therefore, Q holds.

c) First, fill in Q using the postcondition (hint, how does y relate to the `@returns` clause?). Then, use forward reasoning to fill in the missing assertions in the following code:

```
// Computes the square of (x + 1)
// @param x The number to increment and square
// @return (x + 1)^2
public static int nextSquare(int x) {
  int y = x * x;
  {{ ____________________________________ }}
  y = y + (2 * x);
  {{ ____________________________________ }}
  y = y + 1;
  {{ P: _________________________________ }}
  {{ Q: _________________________________ }}
  return y;
}
```

```
public static int nextSquare(int x) {
  int y = x * x;
  {{ y = x^2 }}
  y = y + (2 * x);
  {{ y - 2x = x^2 }}
  y = y + 1;
  {{ P: y - 2x - 1 = x^2 }} <--> {{ y = x^2 + 2x + 1}}
  {{ Q: y = (x + 1)^2}}
  return y;
}
```

d) Show that the code is correct by explaining how P implies the postcondition.

nextSquare returns y. We can see that $y = (x+1)^2$ as

$$\begin{aligned} y &= x^2 + 2x + 1 \quad \text{since } y = x^2 + 2x + 1 \\ &= (x+1)^2 \end{aligned}$$

Alternate solution: We know that $y = x^2 + 2x + 1$ which we can rewrite using algebra that $y = (x+1)^2$. Therefore, the postcondition holds.

Task 2 – Does a Duck Say "Back"?

In this problem, you will practice proving correctness of straight-line code using **backward reasoning**.

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

```
{{ P: x < w + 1 and w > 0 }}
{{ Q: ____________________________________ }}
y = 4 * w;
{{ ____________________________________ }}
x = x * 2;
{{ ____________________________________ }}
z = x - 8;
{{ z < y }}
```

```
{{ P: x < w + 1 and w > 0 }}
{{ Q: 2x - 8 < 4w }}  <-->  {{ 2x < 4w + 8 }}  <-->  {{ x < 2w + 4 }}
y = 4 * w;
{{ 2x - 8 < y }}
x = x * 2;
{{ x - 8 < y }}
z = x - 8;
{{ z < y }}
```

Alternate solution: We know that x < w + 1. Since w > 0, we know that w < 2w, and w < 2w + 3. Thus, x < w + 1 < 2w + 4.

b) Show that the code is correct by explaining how P implies Q.

We can see that Q holds since

$$\begin{aligned} x &< w + 1 \\ &< 2w + 1 \quad \text{since } w > 0 \\ &< 2w + 4 \end{aligned}$$

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

```
// Performs a simple calculation on c
// @param c The input integer
// @requires c >= 0
// @returns an integer >= c
public static int calculation(int c) {
 {{ P: c >= 0 }}
 {{ Q: ____________________________________ }}
 b = 2 * c;
```

```
 {{ ____________________________________ }}
 c = c - 1;
 {{ ____________________________________ }}
 a = b + 1;
 {{ a >= c }}
 return a;
}
```

```
public static int calculation(int c) {
 {{ P: c >= 0 }}
 {{ Q: 2c + 1 >= c - 1 }}  <-->  {{ c >= -2 }}
 b = 2 * c;
 {{ b + 1 >= c - 1 }}
 c = c - 1;
 {{ b + 1 >= c }}
 a = b + 1;
  {{ a >= c }}
 return a;
}
```

d) Show that the code is correct by explaining how the precondition implies Q.

We can see that Q holds since $c \geqslant 0 \geqslant -2$.

Task 3 – Nothing To Be If-ed At

Use forward reasoning to fill in the assertions. Explain how what we know at the end of the conditional implies the post condition.

```
// Calculates a safe substring length starting from a given index.
// @param start The starting index of the substring
// @param desired The desired substring length
// @param len The total length of the original string
// @requires 0 <= start < len and desired >= 0
// @return A window size such that start + window <= len
public static int calcWindow(int start, int desired, int len) {
  {{ ____________________________________ }}
  int window;
  if (start + desired <= len) {
    {{ ____________________________________________________________ }}
    window = desired;
    {{ P1: ________________________________________________________ }}
  } else {
    {{ ____________________________________________________________ }}
    window = len - start;
    {{ P2: ________________________________________________________ }}
  }
 {{ ____________________________________ }}
  return window;
}
```

```
public static int calcWindow(int start, int desired, int len) {
  {{ 0 <= start < len and desired >= 0 }}
  int window;
  if (start + desired <= len) {
    {{ 0 <= start < len and desired >= 0 and start + desired <= len }}
    window = desired;
    {{ P1: 0 <= start < len and desired >= 0 and start + desired <= len and window = desired }}
  } else {
    {{ 0 <= start < len and desired >= 0 and start + desired > len }}
    window = len - start;
    {{ P2: 0 <= start < len and desired >= 0 and start + desired > len and window = len - start }}
  }
 {{ P1 or P2 }}
  return window;
}
```

The postcondition is start + window <= len and we know that "P1 or P2" is true. We will show that the postcondition holds in general:

Suppose P1 is true. We can see:

$$\begin{aligned} start + window &= start + desired && \text{Since window} = \text{desired} \\ &\leqslant len && \text{Since start} + \text{desired} < \text{length} \end{aligned}$$

Suppose P2 is true. We can see:

$$\begin{aligned} start + window &= start + (len - start) \quad \text{Since window} = \text{len - start} \\ &= len \\ &\leqslant len \end{aligned}$$

Since these cases are exhaustive and we have shown that in either case the postcondition holds, we have proven that the postcondition holds in general.

Task 4 – Everybody Loops

In this problem, we will prove the correctness of a method containing a loop that finds the quotient of x divided by 10, i.e., the *largest* value y such that $10y \leqslant x$. To say that y is the largest such value means that any larger value would not satisfy the inequality, i.e., that $10(y+1) \nleqslant x$.

We denote the initial value of the parameter x at the top of the method by x_0. This is explicitly stated in the precondition as the fact "$x = x_0$" (note this is not strictly necessary - you always know $x = x_0$ until you modify it!). The first two facts of Q are from the spec postcondition, which say that y is the quotient of x_0 divided by 10. The third fact is specific to our implementation, and says that x is the remainder, i.e., the remaining amount not divisible by 10.

This method calculates the quotient without division. Instead, it just uses subtraction. It operates by increasing y and decreasing x each time around. The first part of the invariant says that the distance from x_0 down to $10y$ (i.e., $x_0 - 10y$) is the same as the distance from x down to 0 (i.e., $x - 0 = x$). The second part of the invariant says that x has not moved below 0 (i.e., $x \geqslant 0$).

```
// Computes the integer quotient of x divided by 10
// @param x The numerator
// @requires x >= 0
// @return The largest integer y such that 10 * y <= x_0
public static int divideByTen(int x) {
  {{ x = x_0 and x_0 >= 0 }}
  int y = 0;
  {{ P1: _____ }}
  {{ Inv: x_0 - 10y = x and x >= 0 }}
  while (x >= 10) {
    {{ _____ }}
    y = y + 1;
    {{ _____ }}
    x = x - 10;
    {{ P3: _____ }}
    {{ Q2: _____ }}
  }
  {{ P2: _____ }}
  {{ Q1: 10y <= x_0 and x_0 < 10(y+1) and x = x_0 - 10y }}
  return y;
}
```

a) Fill in P1, then show that the invariant is true when we get to the top of the loop the first time.

Forward reasoning tells us that P1: $x = x_0$, $x_0 \geqslant 0$, and $y = 0$. The first part of the invariant holds since

$$\begin{aligned} x_0 - 10y &= x - 10y && \text{since } x = x_0 \\ &= x && \text{since } y = 0 \end{aligned}$$

The second fact holds since $x = x_0 \geqslant 0$

b) Fill in P2, then show that Q1 holds when we exit the loop.

When we exit the loop, we know that P2: inv ($x_0 - 10y = x$, $x \geqslant 0$), and $x < 10$. The first part of the postcondition holds since

$$\begin{aligned} 10y &= x_0 - x && \text{since } x_0 - 10y = x \\ &\leqslant x_0 && \text{since } x \geqslant 0 \end{aligned}$$

the second part of the postcondition holds since

$$\begin{aligned} x_0 &= x + 10y && \text{since } x_0 - 10y = x \\ &< 10 + 10y && \text{since } x < 10 \\ &= 10(y+1) \end{aligned}$$

and the third part is a restatement of the first fact from the invariant. Because the method returns `y`, and Q implies the required bounds for the quotient, the code correctly satisfies the `@return` specification.

c) Fill in Q2 (Hint: what do we know at the end of a loop?). Then, forward reason to P3. Show that P3 implies Q2, proving that the body of the loop is correct.

```
{{ x_0 - 10y = x and x >= 0 and x >= 10 }}
y = y + 1;
{{ x_0 - 10(y - 1) = x and x >= 0 and x >= 10 }}
x = x - 10;
{{ P3: x_0 - 10(y - 1) = (x + 10) and (x + 10) >= 0 and (x + 10) >= 10 }}
{{ Q2: Inv: x_0 - 10y = x and x >= 0 }}
```

We must show that $P3$ implies $x_0 - 10y = x$ and $x \geqslant 0$:
From $P3$, we know $x_0 - 10(y-1) = x + 10$. Simplifying, we get:

$$\begin{aligned} x_0 - 10y + 10 &= x + 10 \\ x_0 - 10y &= x \end{aligned}$$

Also from $P3$, we know $x + 10 \geqslant 10$. Subtracting 10 from both sides gives us $x \geqslant 0$. Thus, the invariant holds.