Questions on specification, Design by Contract, Spec#

David Lightfoot, October 2011

1)

```
static int What0(int x)
  requires x >= 0;
  ensures result * result <= x && x < (result+1)*(result+1);
```

a) What is the value of *What0(26)*?

b) What does the precondition mean?

c) Why do we have that particular pre-condition?

d) What does the post-condition mean?

e) What does *What0* do?

2)

What does the following method do?

```
static void What1 (int [] a)
  requires a != null
ensures forall {int i in (0:a.Length); a[i] == 0};
```

Note: in Spec#, int k in (m: n) means m <= k < n

3)

```
static float AverageLength(string[] s)
 requires ??;
 ensures result = (float) (sum {int k in (0: s.Length); s[k].Length()}) / s.Length;
```

a) What does *AverageLength* do?

b) What should its pre-condition be? (Pay particular attention to possible null values).

c) How could you rewrite *AverageLength* in *Spec#* to make use of facilities for protection against null values?

4)

```
static int Min(int[] a)
  requires a != null;
 ensures forall {int i in (0:a.Length); min <= a[i]};
 { int min = 0;
  for (int j = 0; j != a.Length; j++)
    if (a[j] < min) min = a[j];
  return min;
}
```

a) Does the implementation satisfy the specification of *Min*?

b) If not, how would you fix it?

c) Does the implementation that consists simply of the body

return MIN_VALUE; // the smallest value of type int

satisfy the specification above?

d) Write an improved specification.

5)

```
static int What2(int[] a, int x)
 requires a != null;
 ensures        (0 <= result && result < a.Length &&
                (exists {int k in (0: a.Length); a[k] == x} && a[result] == x) ||
                (!exists {int k in (0: a.Length); a[k] == x} && result == -1);
```

a) What does the pre-condition of *What2* mean?

b) What does the post-condition of *What2* mean?

Given the declaration:

int[] a= {15, 20, 19, 30, 25, 19, 22, -4};

c) What is the value of *What1(a, 40)*?

d) What is the value of *What1(a, 19)*?

6)

```
static int What3(int[] a, int x)
  requires a != null;
  ensures      (0 <= result && result < a.Length &&
                exists {int k in (0: a.Length); a[k] == x} &&
                forall {int j in (0: result); a[j] != x } && a[result] == x) ||
        (!exists {int k in (0: a.Length); a[k] == x} && result == -1);
```

What3 is the same as *What2* but with an extra term in the post-condition:

```
forall {int j in (0: result); a[j] != x }
```

a) What does this additional term mean?

b) Which is easier to implement, *What1* or *What2*?

7)

```
static int What4(int[] a, int x)
  requires a != null && forall {int i in (0: a.Length-1); a[i] <= a[i+1]};
  ensures
        (0 <= result && result < a.Length &&
        exists {int k in (0: a.Length); a[k] == x} &&
        forall {int j in (0: result); a[j] != x } && a[result] == x) ||
        (!exists {int k in (0: a.Length); a[k] == x} && result == -1);
```

What4 is the same as *What3* but with an extra term in the pre-condition:

```
forall {int i in (0: a.Length-1); a[i] <= a[i+1]};
```

a) What does this additional term mean?

b) How does the presence of this extra term affect a possible implementation?

Assertions in Java

We can simulate the effect of *requires* and *ensures* in languages that have assertion handling. Eiffel has a similar feature, with the same keywords. In Java we can use the built-in method:

```
assert Boolean-expression;
```

or

assert Boolean-expression: string;

When the program is run if the Boolean expression is true when the assert method is executed then nothing happens. If it is false a message is issued, including the string used in the method call and the program halts. Assertions are very good documentation because they assert what you believe should be true at a particular point in the program. if your assertion is incorrect then you soon get to know about it and can fix the program.

We can simulate the effect of *requires* and *ensures* in languages implementations that do not have those features:

Where you have *requires pre;*, include *assert pre;* as the first statement of the methods.

Where you have *ensures post;*, put *assert post;* just before the return statement or the textual end of the method.

Note however, that there is no provision for quantifiers, such as *forall*, *exists*, *sum*, so we will need to write our own methods to simulate the effect of these.

Enabling assertion handling in *Netbeans*:

By default assertion handling is turned off and your calls to *assert* are ignored. To turn on assertion handling:

- Select menu File
- Select menu item Project Properties
- Select category node Run
- Select text field VM Options
- Type -ea or -enableassertions

You can check that assertion checking is working by running the

a program with assert false; as the first line of the main method.

8)

static boolean isSmallest (int []a,int min)

that returns true if and only if min is the smallest value in a.

a) Write a specification for *isSmallest*.

b) Write a Java implementation of *isSmallest*.

9)

Use your method *isSmallest* to write a Java implementation of *Min* using Java assertions to achieve the effect of the *requires* and *ensures* of your corrected specification of *Min*.