Assignment 1 CSSE3100/7100 Reasoning about Programs

Personal feedback ***Student's Name***

Question 1 [4 marks]

A pair of integers, (x, y), where x ≤ y, represents a line segment containing all points from x up to, but not including, y (i.e., all integers z such that x ≤ z < y).

For each of the Dafny methods described below, write a method signature and formal specification without using quantifiers (forall and exists). You may define functions to which your specifications refer. You do not need to provide specifications for your functions.

(a) A method to determine whether one line segment (x,y) contains another (x',y'), i.e., all points of (x',y') lie within (x,y).

(b) A method to determine whether one line segment is disjoint from another, where two line segments are disjoint if they have no points in common.

(c) A method to find the smallest line segment which contains two given line segments.

Feedback:

(a) Precondition (0.5 marks): *Comment*

Your mark: *0.5*

Postcondition (0.5 marks): *Comment*

Your mark: *0.5*

A sample solution is shown below. Inputs have been restricted in the preconditions to represent non-empty line segments.

```
method contains(x: int, y: int, x': int, y': int) returns (b: bool)
    requires  x < y && x' < y'
    ensures  b <==> x <= x' && y'<= y
```

(b) Precondition (0.5 marks): *Comment*

Your mark: *0.5*

Postcondition (0.5 marks): *Comment*

Your mark: *0.5*

A sample solution is shown below.

```
method disjoint(x: int, y: int, x': int, y': int) returns (b: bool)
    requires x < y && x' < y'
    ensures b <==> y <= x' || y' <= x
```

(c) Precondition (0.5 marks): *Comment*

Your mark: *0.5*

Postcondition (1.5 marks): *0.5 for max and 0.5 for min (students may have inlined these definitions, and that's OK), 0.5 for the first postcondition*

Your mark: *1.5*

A sample solution is shown below.

```
function min(x: int, y: int): int {
      if x <= y then x else y }

function max(x: int, y: int): int {
      if x <= y then y else x }

 method smallestContainer(x: int, y: int, x': int, y': int) returns (a: int, b: int)
     requires x < y && x' < y'
     ensures a == min(x, x') && b == max(y, y') && a <= b
```

The final conjunct, a ≤ b, is not strictly necessary (it follows from the precondition), but can be useful for documenting your intention, and for allowing a person or tool to check that your precondition is sufficient.

Question 2 [4 marks]

Your friend claims that the method below always terminates and returns x * y, provided x and y are non-negative. Prove that your friend is either right or wrong. To do so, complete the specification below and use weakest precondition reasoning to show that the implementation satisfies the specification.

If the implementation is not correct, explain the cases when it will fail to compute x * y.

```
method mult(x: int, y: int) returns (r: int)
      requires …
```

```
        ensures ...
        decreases ...
{
        if x < y {
                r := mult(y, x);
        } else if y !=0 {
                r := mult(x, y-1);
                r := r + x;
        } else {
                r := 0;
        }
}
```

Precondition (0.5 marks): *Comment*

Your mark: *0.5*

Postcondition (0.5 marks): *Comment*

Your mark: *0.5*

Termination metric (0.5 marks): *Comment*

Your mark: *0.5*

Weakest precondition proof (2.5 marks) : *Comment*

Your mark: *2.5*

A sample solution is shown below (0.5 marks was given for each line indicated with ←).

```
method mult(x: int, y: int) returns (r: int)
        requires  x >=0 && y >= 0
        ensures  r == x * y
        decreases  y, x
{
        {  x >= 0 && y >= 0  }              ←
        // since x >= y && y >= 0 implies x >= 0, and again using the law
        // (A && B) || (A && C)  <==>  (A && (B || C))
        { (x >= y && y >= 0) || (x >= 0 && y >= 0 && x < y) || (x >= 0 && y >= 0)  }
        // using the law, (A && B) || (A && C)  <==>  (A && (B || C)), twice
        {  false || (x >= y && y == 0) || (x >=y && y >0) ||
           (x >= 0 && y >= 0 && x < y) || (x >= 0 && y ==0) ||  (x >= 0 && y > 0)  }
        {  (x >= y && x < y) || (x >= y && y == 0) || (x >=y && y >0) ||
           (x >= 0 && y >= 0 && x < y) || (x >= 0 && y >= 0 && y ==0) ||
```

```
    (x >= 0 && y >= 0 && y > 0)  }
{  (x >= y || (x > = 0 && y >= 0)) && (x < y || y == 0 || (x >= 0 && y > 0))  }
{  (x < y ==> x  >= 0 && y >= 0 ) && (x >= y ==> y == 0 || (x >= 0 && y > 0))  }
if x < y {
        {  x  >= 0 && y >= 0  }          ←
        { y >= 0 && x >= 0 && forall r' :: r ' == y * x ==> r' = x * y  }
        r := mult(y, x);
        {  r == x * y  }
{  y == 0 || (x >= 0 && y > 0)  }        ←
{  (y != 0 ==> x >= 0 && y > 0) && (y == 0 ==> x * y == 0)  }
} else if y !=0 {
        {  x >= 0 && y > 0  }   ←
        { x >= 0 && y-1 >= 0 && x * (y-1) + x == x * y  }
        { x >= 0 && y-1 >= 0 && forall r' :: r ' == x * (y-1) ==> r' + x == x * y  }
        r := mult(x, y-1);
        {  r + x == x * y  }
        r := r + x;
        {  r == x * y  }
} else {
        {  x * y == 0  }          ←
        r := 0;
        {  r == x * y  }
}
{  r == x * y  }
}
```

Note that x, y would not work for the termination metric. When x < y, there is a call to mult(y, x) in which x, y increases.

Question 3 [2 marks]

A particular brand of breakfast cereal costs *b* dollars per box. In each box there is one token. For *t* tokens, you get a free box of cereal (also with a token in it). Write a method signature and specification that determines how many boxes of cereal you can get with *m* dollars taking into account free boxes you get with tokens. *m*, *b* and *t* should be parameters to the method.

Precondition (0.5 marks): *Comment*

Your mark: *0.5*

Postcondition (1.5 marks): *Give 0.5 for m/b and 0.5 for each term of the recursive function, or 1 for the second term of the non-recursive solution.*

Your mark: *1.5*

This problem was intentionally a little more challenging. Here are 2 sample solutions. The first uses recursion, the second does not.

Sample solution 1:

```
function boxes_from_tokens(tokens: int, t: int): int {
        if tokens < t then 0 else tokens/t + boxes_from_tokens(tokens/t + tokens % t, t)
}

method num_boxes(m: int, b:int, t: int) returns (r: int)
          requires  b > 0 && t > 1
          ensures   r == m/b + boxes_from_tokens(m/b, t)
```

This solution directly captures the recursive nature of the problem. Although the question does not stipulate that *b* has to be greater than 0, if it were not we would get an undefined value in our specification due to dividing by *b*.

Sample solution 2:

```
method num_boxes(m: int, b:int, t: int) returns (r: int)
          requires  b > 0 && t > 1
          ensures   r == m/b + r/t
```

In this solution, we use the fact that r can appear anywhere in the predicate, and so can be defined in terms of itself.

Total Mark: *10*