Name: _____

CSSE 373—Formal Methods in Specification and Design

Exam 2, May 7, 2009

This exam is open book and open notes. You may also use your laptop if you wish to review past homework, slides, or reference manuals, though you should budget your time very carefully. You may *only* use your laptop to access data on your local hard drive or directly accessible from the course ANGEL or web pages. You *may not* use Eclipse or the ESC/Java checker.

Problem	**Poss. Pts.**	**Earned**
1	15	_____
2	10	_____
3	10	_____
4	10	_____
5	15	_____
6	20	_____
7	20	_____
Total	100	_____

1. (15 points) Suppose we're creating a class to represent rational numbers. A portion of the class is given below. Fill in the invariants so that the equalTo method behaves correctly. *Note*: This is equivalent to specifying the invariants so that every rational number has a unique representation.

Example: to represent the number 2.25 using Rational, the field values would be:

whole:	2
num:	1
den:	4

```
public class Rational {
   private /*@ spec_public @*/ int whole;  // the non-fractional portion
   private /*@ spec_public @*/ int num;    // the numerator of the fraction
   private /*@ spec_public @*/ int den;    // the denominator of the fraction

   // Several lines are given for your answer. You might not need them all.

   //@ invariant _____

   //@ invariant _____

   //@ invariant _____

   //@ invariant _____

   //@ invariant _____

   //@ invariant _____

   public Rational(int whole, int num, int den) {
      ...
   }

   ...

   //@ requires other != null;
   //@ assignable /nothing;
   /*@ ensures \result <==>
     @          (this.whole * this.den + this.num) * other.den
     @            == (other.whole * other.den + other.num) * this.den;
     @*/
   public boolean equalTo(Rational other) {
      return this.whole == other.whole
            && this.num == other.num
            && this.den == other.den;
   }
}
```

2. (10 points) Find the weakest pre-condition:

```
  //@ assert _____________________________________
i = i + 1;
  //@ assert i > 0;
```

3. (10 points) Find the weakest pre-condition:

```
  //@ assert _____________________________________
a[i] = 1;
  //@ assert a[i] == a[j];
```

4. (10 points) Find the weakest pre-condition:

```
  //@ assert _____________________________________
a = a + b;
  //@ assert a == 10 && b == 13;
```

5. (15 points) Find the weakest pre-condition. Show your work.

```
  //@ assert _____________________________________
if (x > y) {

    //@ assert _____________________________________
  x = x - y + 1;

    //@ assert _____________________________________
} else {

    //@ assert _____________________________________
  x = 1;

    //@ assert _____________________________________
}
  //@ assert x > 0;
```

6. (20 points) Assume that the procedure cd has the following specification:

```
//@ requires x > 0;
//@ assignable \nothing;
//@ ensures \result == 2 * x;
int cd(int x);
```

Prove that the following implementation of cd satisfies the specification. Show your work.

```
int cd(int x) {

   //@ assert _____
  int r;
  if (x == 1) {

     //@ assert _____
    r = 2;

     //@ assert _____
  } else {

     //@ assert _____

     //@ assert _____

     //@ assert _____
    r = cd(x - 1);

     //@ assert _____

     //@ assert _____
    r = r + 2;

     //@ assert _____
  }

   //@ assert _____
  return r;
}
```

7. (20 points) Show that the following program satisfies its pre- and post-conditions. Some of the steps are completed for you. Show your work.

```
  //@ assert a > 0;
  //@ assert a >= 1 && 1 == 1;
  //@ assert a >= 1 && 1 == (\product int k; a+1 <= k && k <= a; k);
r = 1;
  //@ assert a >= 1 && r == (\product int k; a+1 <= k && k <= a; k);
b = a;
  //@ maintaining b >= 1;
  //@ maintaining r == (\product int k; b+1 <= k && k <= a; k);
  //@ decreasing b;
  //@ assert  b >= 1 && r == (\product int k; b+1 <= k && k <= a; k);
while (b > 1) {

    //@ assert _____

    //@ assert _____

    //@ assert _____
  r = r * b;

    //@ assert _____
  b = b - 1;

    //@ assert _____
}

    //@ assert _____
    //@ assert r == (\product int k; 1 <= k && k <= a; k);
```