CS 6340 - Spring 2016
Assignment 1 (100 points)
Due on: midnight (anywhere on Earth), Jan 24, 2016

Objective: The goal of this assignment is to become familiar with a technique and tool for formally reasoning about partial correctness properties of programs. In particular, we will use the Dafny program verifier from Microsoft Research.

Resources:

1. Dafny interactive tool: `http://rise4fun.com/Dafny`
2. Dafny quick reference: `http://research.microsoft.com/en-us/projects/dafny/reference.aspx`
3. Dafny homepage: `http://research.microsoft.com/en-us/projects/dafny/`

Setup: Try each of the below questions in the Dafny interactive tool at `http://rise4fun.com/Dafny`. The interactive tool allows to create a Permalink (a URL whose content is no longer modifiable) to the final Dafny program you enter. Submit a single text document on T-Square named dafny.txt containing a Permalink of each problem below. Example submission:

1. `http://rise4fun.com/Dafny/W0n`
2. `http://rise4fun.com/Dafny/Iwht`
3. `http://rise4fun.com/Dafny/lH4d`
4. `http://rise4fun.com/Dafny/pG6j`

Problems:

Problem 1. [**20 points**] The class declared below mimics a Lock class in programming languages like Java and C++. Insert the right **requires** statements to pass Dafny's check. [Also available at `http://rise4fun.com/Dafny/W0n`]

```
class Lock {
    var state:bool;

    constructor init()
    modifies this;
    ensures state == false;
    {
        state := false;
    }


    method acquireLock()
    modifies this;
    ensures state == true;
    {
        state := !state;
    }

    method releaseLock()
    modifies this;
    ensures state == false;
    {
        state := !state;
    }
}
```

Problem 2. [**20 points**] Please insert the right **invariant** and **decreases** statements for the program below to pass Dafny's termination check. [Also available at `http://rise4fun.com/Dafny/Iwht`]

```
method Main() {
    var a:int := 0;
    var b:int := -1;
    var c:int := 0;
    var i:int := 100;
    while (a!=b)
    {
        b := a;
        c := c+1;
        if (c < i) {
            a := a+1;
        }
    }
    print "Eureka";
}
```

Problem 3. [**20 points**] Now let us combine the above two problems together. Insert the right **invariant** and **decreases** statements to make the program below pass Dafny's check. [Also available at `http://rise4fun.com/Dafny/lH4d`]

```
class Lock {
    var state:bool;

    constructor init()
    modifies this;
    ensures state == false;
    {
        state := false;
    }


    method acquireLock()
    modifies this;
    ensures state == true;
    {
        state := !state;
    }

    method releaseLock()
    modifies this;
    ensures state == false;
    {
        state := !state;
    }
}

method Main() {
    var a:int := 0;
    var b:int := -1;
    var c:int := 0;
    var l:Lock := new Lock.init();
    var i:int := 100;
    while (a!=b)
    {
        b := a;
        c := c+1;
        l.acquireLock();
        if(c < i){
            a := a+1;
            l.releaseLock();
        }
    }
    l.releaseLock();
    print "Eureka";
}
```

Bonus Problem. [**40 points**] The following program in Dafny defines the sorted predicate and bubbleSort sorting algorithm. Insert **invariant** statements to pass Dafny's check (the invariant statements for the outer loop are already provided). [Also available at `http://rise4fun.com/Dafny/pG6j`]

```
predicate sorted(a:array<int>, left:int, right:int)
requires a!=null && 0 <= left <= right <= a.Length;
reads a;
{
    forall x:int :: left<=x<right-1==> a[x]<=a[x+1]
}

method bubbleSort(a: array<int>)
requires a != null && a.Length > 1;
modifies a;
ensures sorted(a, 0, a.Length);
{
    var sortedUntil := 0;
    var i := a.Length - 1;
    while (sortedUntil < a.Length)
    invariant 0 <= sortedUntil <= a.Length;
    invariant forall j, k :: 0 <= j < sortedUntil <= k < a.Length ==> a[j] <= a[k];
    invariant sorted(a, 0, sortedUntil);
    {
        i := a.Length - 1;
        while(i > sortedUntil)
        {
            if(a[i] <= a[i - 1])
            {
                a[i - 1], a[i] := a[i], a[i-1];
            }
            i := i - 1;
        }
        sortedUntil := sortedUntil + 1;
    }
}
```