Construction and Verification of Software Course / MIEI / FCT UNL
Midterm Test 20 April 2015

1. Recall that a Hoare triple {P} S {Q} is said valid if whenever S is executed in any state that satisfies assertion P it will necessarily terminate in a state that satisfies assertion Q. For each triple (we use Dafny syntax), say if it is valid or invalid.

```
{ a!=null && 0<=n<=a.Length && 0<=i<n }
if(a[i] > a[i]){x := 0}else{x := 1}
{ x == 1 }

{ n > 0 && n % 2 == 1 && x > y}
while(n>0){z:=x;x:=y;y:=z;n:=n-1}
{ x < y }
```

2. For each Hoare triple (we use Dafny syntax), write the weakest precondition or the strongest postcondition you can think of that makes the triple valid.

```
{                                                        }
y := x*x
{ y <= 2 }

{ |x – y| <= 2 }
if(x>y){z:=x-y} else {z:=y-x}
{                                                        }

{                                                        }
y := 2*i; a[y]:=a[i];
{ a[j] == 4 }
```

3. For the following code, add the pre-conditions, post-conditions and loop invariants that will make a most safe and precise specification.

```
method mix(a:array<int>, b:array<int>, m:array<int>, n:int)
modifies m;
requires [                                          ];
requires [                                          ];
requires [                                          ];
ensures  [                                          ];
ensures  [                                          ];
{
   var i : int := 0;
   var j : int := 0;
   while(i<n)
   invariant [                                      ];
   invariant [                                      ];
   invariant [                                      ];
   {
      m[j] := a[i];
      m[j+1] := b[i];
      i := i+1;
      j := j+2;
   }
}
```

4. Consider the following specification of an abstract data type LINEEDIT for representing a line of text and some simple editing operations on it.

NOTE: Answer to be provided in the next page.

A LINEEDIT ADT manages a sequence of characters with at most M=256 elements and length len such that 0 <= len <= M. There is also a current cursor position cp such that 0 <= cp <= len). Initially, there is no text in the line, and the cursor is at position 0. Operations (expressed informally) are the following. We deliberately omit preconditions, which you should provide.

```
{ ???? }
      method typeKey(ch:char)
```
{ *line is the same except ch is inserted at cursor position, and cursor moves one position right* }

```
{ ???? }
      method select(l:int)
```
{ *changes the cursor position to l, but does not change the line content. The new position must be valid within the current line* }

```
{ ???? }
      method backSpace()
```
{ *line is the same except char at the left of the cursor is deleted. If the cursor is at the start of the line, the operation does nothing* }

```
{ ???? }
      method charAt(p:int) returns (ch:char)
```
{ *returns the char at p position* }

```
{ ???? }
      method lineLen() returns (l:int)
```
{ *returns the length of the current line* }

Example:

Starting from a new line `l=new LINEEDIT();` and after executing `l.typeKey('h'); l.typeKey('o'); l.backSpace(); l.typeKey('i'); l.select(0); l.typeKey('!');` we would obtain `l.charAt(0)=='!'`, `l.charAt(1)=='h'`, `l.charAt(2)=='i'` and `l.lineLen()==3`.

4.1. Define the representation type for the LINEEDIT ADT and define the representation invariant as a boolean function RepInv(). Recall that the representation invariant characterizes the concrete states that are sound and represent well-defined abstract states.

4.2. Provide full Dafny code for all operations of the LINEEDIT implementation, ensuring that the representation invariant is preserved by all operations. Provide loop invariants for any loops you might need.

4.3. Provide appropriate postconditions for all operations.