Construction and Verification of Software – 2019/2020

Self Assessment Test 15 April, 2020

Notes: This is a self-assessment test. It is designed to be closed book and for a duration of 1h30m. There are 4 open answer questions.

Version: A

Name:_____Number:_____

Q-1 Given the Hoare triple

$\{P\}$ `t := x; x := y; y := t` $\{Q\}$

Define the weakest pre-condition and strongest post-conditions conditions P and Q, that make the triple valid.

Q-2 Taking into account what you learnt about Hoare Logic and the following Hoare triple:

$\{A\}$ `if (x%2 == 0) {z := z * 2;} else {y := y * 2;}` $\{z\%2 = 0 \wedge y\%2 = 0\}$

Define the weakest precondition possible A that makes the Hoare triple above correct.

Q-3 Considering the program with a placeholder

```
i := 0; n := 0;
while (i < 10)
[                                                    ]
{ n := (n + 1)%2; }
```

Define the loop invariant that established the strongest post-conditions for the program fragment.

Q-4 Select the Hoare triples, expressed here as a Dafny methods, that are **not** valid.

A -
```
method m(a:array<int>, n:int) returns (x:int)
  requires 1 < n < a.Length;
  ensures true
  {  return a[n] * a[n-1]; }
```

B -
```
method m(a:array<int>) returns (b:bool)
  requires 0 < a.Length
  ensures b ==> a[0] == 0
  { return a[0] == 0; }
```

C -
```
method m(y:int, w:int) returns (x:int)
  requires y > 0 && -y <= w <= 0
  ensures x > 0
  { x := y + w; }
```

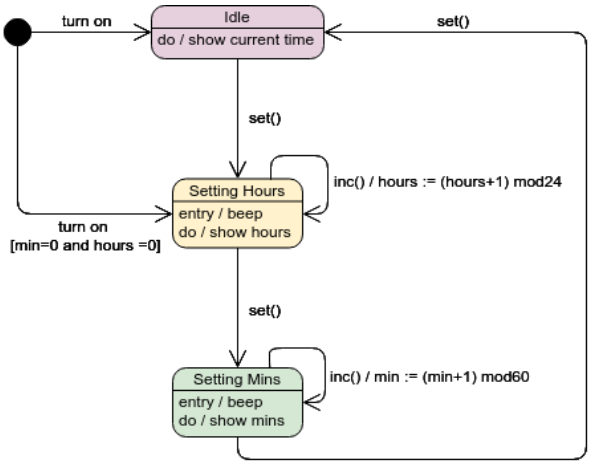
D -
```
method m(x:int) returns (y:int)
  requires x == 1 && x == 2
  ensures y > 0
  { y := -1; }
```

E -
```
method m(y:int, w:int) returns (x:int)
  requires y > 0 && w < 0
  ensures x > 0
  { x := y + w; }
```

Q-5 Complete the code below with the strongest post-conditions, the weakest pre-conditions possible, and the needed invariants so that Dafny verifies the code without errors.

```
function count(a:array<int>,p:int, i:int):int
  requires 0 <= i <= a.Length
  reads a
  decreases i
{ if i == 0 then 0 else if a[i-1] == p then count(a,p,i-1) + 1 else count(a,p,i-1) }

method Count(a:array<int>, x:int) returns (s:int)

  ensures [                                                              ]

{
  var i := 0;
  s := 0;
  while i < a.Length
    decreases a.Length - i

    invariant [                                                          ]

    invariant [                                                          ]

  {
    if a[i] == x
      { s := s + 1; }
    i := i + 1;
  }
}
```

Q-6 Consider an ADT representing the control mechanism for a Digital Clock. It controls the configuration interface of the device. Follow the state diagram on the right to complete the Dafny code below such that it represents all state transitions of the object. Complete the specification of the class by adding field declarations and functions that help define all needed TypeStates and conditions.




```
class DigitalClock {
  var state: int;
  var hours:int;
  var minutes: int;

  function method Idle():bool reads `state { state == 0 }
  function method SettingHours():bool reads `state { state == 1 }
  function method SettingMin():bool reads `state { state == 2 }

  constructor()
  ensures Idle()
  { state := 0; hours := 0; minutes := 0; }

  method Set()
  requires [                                                    ]

  ensures [                                                     ]

  ensures [                                                     ]

  ensures [                                                     ]

  modifies `state
  {
    state := (state + 1)%3;
  }

  method inc()
  requires [                                                    ]

  ensures [                                                     ]

  ensures [                                                     ]

  modifies `hours, `minutes
  {
    if SettingHours()
    {
      hours := (hours + 1)%24;
    } else {
      minutes := (minutes + 1)%60;
    }
  }
}
```