TÖL212M Rökstudd Forritun - Einstaklingsverkefni 7

Andri Fannar Kristjánsson

2. mars 2025

Einstaklingsverkefni 7

1

Sækið skrána `E7-skeleton.dfy` og vistið hana hjá ykkur en breytið nafni hennar í E7.dfy. Klárið að forrita klasann í skránni.

1.1 Svar:

Hér fyrir neðan má sjá kóðann þar sem klasarnir hafa verið forritaðir. Dafny samþykkir þessa útgáfu, þótt `Tio.run` geri það ekki. Einnig er hægt að skoða kóðann á þessari slóð: https://tinyurl.com/4y5m4cs4.

```
// Author of question:  Snorri Agnarsson, snorri@hi.is

// Author of solution:       Andri Fannar Kristjánsson, afk6@hi.is
// Permalink of solution:    https://tinyurl.com/4y5m4cs4

// Klárið að forrita klasann IntStackChain.

// Finish programming the class IntStackChain.

trait IntStack
  {
  ghost var ghostseq: seq<int>
  ghost var Repr: set<object>

  ghost predicate Valid()
    reads this, Repr

  predicate IsEmpty()
    reads this, Repr
    requires Valid()
    ensures IsEmpty() <==> ghostseq==[]

  method Push( x: int )
    modifies this, Repr
    requires Valid()
    ensures Valid() && fresh(Repr-old(Repr))
    ensures ghostseq == old(ghostseq)+[x]

  method Pop() returns ( x: int )
    modifies this, Repr
    requires Valid()
    requires ghostseq != []
    ensures Valid() && fresh(Repr-old(Repr))
    ensures ghostseq == old(ghostseq[..|ghostseq|-1])
    ensures x == old(ghostseq[|ghostseq|-1])
```

```
}

datatype Chain = Nil | Cons(int,Chain)

function SeqOfChain( x: Chain ): seq<int>
{
  match x
  case Nil => []
  case Cons(h,t) => [h]+SeqOfChain(t)
}

predicate IsReverse( x: seq<int>, y: seq<int> )
{
  |x| == |y| &&
  forall i | 0 <= i < |x| :: x[i] == y[|x|-1-i]
}

function Head( c: Chain ): int
  requires c != Nil
{
  match c
  case Cons(h,t) => h
}

function Tail( c: Chain ): Chain
  requires c != Nil
{
  match c
  case Cons(h,t) => t
}

class IntStackChain extends IntStack
  {
  var c: Chain

  ghost predicate Valid()
    reads this
  {
    // Hér vantar skilgreiningu á fastayrðingu gagna.
    // Notið IsReverse og SeqOfChain til að skilgreina
    // hvenær hlaðinn er í löglegu ástandi.

    // Here a definition of the data invariant is missing.
    // Use IsReverse and SeqOfChain to define when the
    // stack is in a valid state.
    IsReverse(ghostseq, SeqOfChain(c))
  }

  constructor()
    ensures Valid() && fresh(Repr-{this})
    ensures ghostseq == []
  {
    c := Nil;
    Repr := {};
    ghostseq := [];
  }

  predicate IsEmpty()
```

```
    reads this
    requires Valid()
    ensures IsEmpty() <==> ghostseq==[]
  {
    c == Nil
  }

  method Push( x: int )
    modifies this
    requires Valid()
    ensures Valid()
    ensures Repr == old(Repr)
    ensures ghostseq == old(ghostseq)+[x]
  {
    // Hér vantar forritstexta.
    // Segð á sniðinu Cons(h,t) er gagnleg hér.

    // Code is missing here.
    // An expression of the form Cons(h,t) is useful here.
    c := Cons(x, c);
    ghostseq := ghostseq + [x];
  }

  method Pop() returns ( x: int )
    modifies this
    requires Valid()
    requires ghostseq != []
    ensures Valid()
    ensures Repr == old(Repr)
    ensures ghostseq == old(ghostseq[..|ghostseq|-1])
    ensures x == old(ghostseq[|ghostseq|-1])
  {
    // Hér vantar forritstexta.
    // Föllin Head og Tail eru gagnleg hér.

    // Code is missing here.
    // The functions Head and Tail are useful here.
    x := Head(c);
    c := Tail(c);
    ghostseq := ghostseq[..|ghostseq|-1];

  }
}

method Factory() returns ( s: IntStack )
  ensures fresh(s)
  ensures fresh(s.Repr)
  ensures s.Valid()
  ensures s.IsEmpty()
{
  s := new IntStackChain();
}

method Main()
{
  var s := [1,2,3];
  var s1 := Factory();
  var s2 := Factory();
```

```
  while s != []
    decreases |s|
    invariant s1.Valid()
    invariant s2.Valid()
    invariant ({s1}+s1.Repr) !! ({s2}+s2.Repr)
    invariant fresh(s1.Repr)
    invariant fresh(s2.Repr)
  {
    s1.Push(s[0]);
    s2.Push(s[0]);
    s := s[1..];
  }
  while !s1.IsEmpty()
    decreases |s1.ghostseq|
    invariant s1.Valid()
    invariant s2.Valid()
    invariant ({s1}+s1.Repr) !! ({s2}+s2.Repr)
    invariant fresh(s1.Repr)
    invariant fresh(s2.Repr)
  {
    var x := s1.Pop();
    print x;
    print " ";
  }
  while !s2.IsEmpty()
    invariant s2.Valid()
    decreases |s2.ghostseq|
    invariant fresh(s2.Repr)
  {
    var x := s2.Pop();
    print x;
    print " ";
  }
}
```