Lab Session 10
Separation Logic and the Verifast tool

Software Verification

Nova School of Science and Technology
Mário Pereira   mjp.pereira@fct.unl.pt

Version of November 11, 2025

1 Exercises

Exercise 1. Install the Verifast tool.

□

Exercise 2. Implement and specify a Counter class that uses two integer Cell fields. For reference, you can use the Java implementation provided in Appendix A.

□

Exercise 3. Extend your Counter class with a *copy constructor*: it takes as an argument an object other of class Counter and assigns the fields of this with the appropriate values.

Do you need to extend, as well, the Cell class?

□

Exercise 4. Write and verify a method that, given an array and an index into the array, returns the element stored in that position.

□

Exercise 5. Write and verify a method that, given an array, an index into the array and a value to place in the array, updates the index of the array to contain the given value. **Hint:** you will need to use the lemma given below, and the list specification function store:

```
lemma void store_take_drop<t>(list<t> xs, int index, t v)
  requires 0 <= index && index < length(xs);
  ensures store(xs, index, v) == append(take(index, xs), cons(v, drop(index + 1, xs)));
{
  switch(xs) {
    case nil:
    case cons(h, t):
      if(index == 0) {
      } else {
        store_take_drop(t, index - 1, v);
      }
  }
}
```

□

Exercise 6. [⋆⋆] Write and verify a method that sums the elements of a given array, iterating over the array from the higher to the lower indices of the array. □

Exercise 7. [⋆⋆⋆] Write and verify a method that returns the minimum element of a (non-empty) array. It will be helpful to factor your implementation into two methods, one that computes the index of the minimum element of the array, starting at a given index, and another that calls this method.

```
  public static int min(int[] a)
    //@ requires a != null &*& array_slice(a, 0, a.length, ?vs) &*& vs != nil;
    //@ ensures ...
  {

    int tmp = indexOfMin(a, 0);
    //@ length_drop(tmp, vs);
    //@ nth_drop(vs, tmp);
    //@ mem_nth(tmp, vs);
    return a[tmp];
  }

  public static int indexOfMin(int[] a, int start)
    //@ requires a != null &*& array_slice(a, start, a.length, ?vs) &*& vs != nil &*& length(
         vs) != 0;
    //@ ensures ...
  {
    ...
  }
```

The following auxiliary definition will likely be helpful:

```
fixpoint boolean forall_le(list<int> vs, int v) {
  switch(vs) {
    case nil: return true;
    case cons(h, t): return v <= h && forall_le(t, v);
  }
}
```

□

A Counter

```
class Cell
{
    public int data;

    public Cell (int n)
    {
        data = n;
    }
}

class Counter
{
    Cell incs;
    Cell decs;

    public Counter ()
    {
        incs = new Cell(0);
        decs = new Cell(0);
    }

    public int getValue()
    {
      return incs.data - decs.data;
    }

    public void inc()
    {
        incs.data = incs.data + 1;
    }

    public void dec()
    {
        decs.data = decs.data + 1;
    }
}
```