Lab Session 6
Loop Invariants, Searching, and Sorting

Software Verification

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

Version of October 14, 2025

Exercise 1. Specify and write an iterative function that computes the maximum of a (non-empty) array of integers, with the following signature:

```
let max_array (arr: array int) : int                                    WhyML
```

You will have to specify the weakest pre-condition and the strongest post-condition for the function. Use quantifiers in the post-condition and the loop invariant.

□

Exercise 2. Are you sure your specification ensures the returned value is the maximum element of the array and not just a supremum of all the elements? You can determine this by checking your specification against the following implementation:

```
let supremum_array (arr: array int) : int                               WhyML
  requires arr.length > 0;
  ensures ...
= let ref sup = arr[0] in
  let ref i = 1 in
  while i < arr.length do
    invariant ...
    ...
  done;
  sup
```

If the function above satisfies your specification then that means your specification can be made more precise. Specifically, you must find a way of asserting (and proving) that the return value of your original method is an element of the array. Go ahead and fix your specification accordingly.

□

Exercise 3. It is likely you wrote your maximum method with a cursor starting at `0`. Try to do it the other way around: compute the maximum element of an array, starting from the last element in the array down to the first.

□

Exercise 4. Write a recursive function `sum n` that computes the sum of all natural numbers between `0` and `n`, starting from `n` (i.e. $n + (n-1) + \cdots + 1$) and then fill out the following function:

```
let sum_backwards (n: int) : int                                        WhyML
  ensures { result = sum(n) }
```

□

Exercise 5. Implement a function called `search` that takes an array of integers and an integer and returns either -1 if the integer is not in the array and the index into the array where the integer can be found. Try to write the strongest post-condition possible. □

Exercise 6. Specify and implement function `fill_k a n k c`. This function returns `true` if and only if the first `c` elements, up to `n`, of array `a` are equal to `k`.

Define the weakest pre-condition and the strongest post-condition possible. Implement the method so that it verifies.

```
let fill_k (a: array int) (n k c: int) : bool                           WhyML
```

□

Exercise 7. Specify and implement the function `contains_sub_string`. This method tests whether or not the array of characters `a` contains the elements of array `b`. If `a` contains `b`, then the method returns the offset of `b` in `a`. If `a` does not contain `b` then the method returns an illegal index (e.g. `-1`).

Define the weakest pre-condition and the strongest post-condition possible. Implement the method so that it verifies.

Hint: you may want to define auxiliary functions and methods (e.g. a method that tests whether the substring starting at a given index of a string is exactly another given string).

```
let contains_sub_string (a b:array char) : int                          WhyML
```

□

Exercise 8. Specify and implement the function `resize`. This method returns a new array whose length is double of the length of the array given as argument (`a`). If the length of the array supplied as an argument is zero, then set the length of the resulting array (`b`) to a constant of your choice.

All the elements of array `a` should be inserted, in the same order, in array `b`.

Define the weakest pre-condition and the strongest post-condition possible. Implement the method so that it verifies.

```
let resize (a: array int) : array int                                   WhyML
```

□

Exercise 9. Specify and implement function `reverse`. This function receives an array `a` and returns a new array (`b`) in which the elements of a appear in the inverse order.

For instance, the inverse of array `a == [0, 1, 5, *, *]`, where `'*'` denotes an uninitialized array position, results in `b == [5, 1, 0, *, *]`.

Define the weakest pre-condition and the strongest post-condition possible. Implement the method so that it verifies.

```
let reverse (a: array int) (n: int) : array int                         WhyML
```

□