# Handout 1
# Sums, Arrays and Lists

Construction and Verification of Software
FCT-NOVA
Bernardo Toninho

28 March, 2023

This handout is due on Saturday, April 15, at 23h59m. The exact details on how to turn in your solution will be made available at a later date, but you will only have to submit a zip file with your Dafny source files.

The handout consists of two verification exercises: one on calculating the ranged sums of array elements (Section 1) and another on the relationship between a functional list and its representation as an array (Section 2).

## 1 Cumulative Sums over Arrays

In many programming scenarios, we are faced with a problem which can be encoded as follows: given a fixed-length sequence of (integer) values, we want to compute some ranged queries over the elements of the sequence. For this particular assignment, our ranged query of choice is a ranged sum (i.e., the sum of all elements within a given range in the sequence).

(a) Define a Dafny specification function called `sum` that takes as arguments an array of integers `a` and two indices into the array, `i` and `j`, that calculates the sum over the interval $[\mathtt{a}_\mathtt{i}, \mathtt{a}_\mathtt{j}[$, with $\mathtt{i} \leq \mathtt{j}$. This function will recur over the array either "from the start" (i.e., recursive calls increase the argument `i`), or "from the end" (i.e., recursive calls decrease the argument `j`). **Note:** You will need to add to the function signature a specification of the form `reads a`, to indicate that the function can read from the memory denoted by the array. You will also likely reduce future pain if you constrain (i.e., `require`) `i` and `j` such that both `sum(a,a.Length,a.Length)` and `sum(a,0,0)` are valid function calls.

(b) In Dafny, implement a `query` method that takes as arguments an array of integers `a` and two indices into the array, `i` and `j`, that calculates the sum over the interval $[\mathtt{a}_\mathtt{i}, \mathtt{a}_\mathtt{j}[$, with $\mathtt{i} \leq \mathtt{j}$. Your method **must** be imperative (i.e., use mutable variables and loops) and **cannot** be recursive. Moreover, your method must `ensure` its result value coincides with the corresponding call of `sum` from point (a).

**Hint:** Note that you can implement this method by traversing the array from lower to higher indices or vice-versa. As usual, coming up with the right loop invariant will be your biggest challenge. While there is more than one correct loop invariant, your particular choices of invariant, implementation and recursive function will affect how easy (or

hard) it is to prove the invariant. Depending on these choices, you may find that Dafny is unable to immediately prove that adding an element to your running sum is the same as adding that element to `sum` over the smaller (by 1) interval. If this is the case, you will need a **`lemma`** (which you will have to state and ensure that Dafny can prove) that relates the value of `sum(a,i,j)` and adding a value from the end (respectively, the start) of the array to the result of summing the elements over the smaller interval.

(c) You will now implement a version of `query` that runs in $O(1)$ time by presupposing that we have pre-computed all *prefix cumulative* sums over the given array. Given an array $[1, 10, 3, -4, 5]$, its prefix sums are given by the array $s = [0, 1, 11, 14, 10, 15]$. That is, each element of array `s` corresponds to the sum of all elements of the original array up to a given index. Note that $s_0$ corresponds to the sum of no indices of the array (i.e., the index interval $[0, 0[$), $s_1$ to the sum of the index interval $[0, 1[$, $s_2$ to the sum of the index interval $[0, 2[$, and so on, up to the length of the array.

The goal is to implement a method that, given a sequence of prefix sums (as an array) `c` for a given array `a`, and indices `i` and `j` as before, returns the sum of all elements in the original array in the index interval $[i, j[$ in $O(1)$ time. The signature of the method should be (you will need fill in the blanks):

```
method queryFast(a: array<int>, c: array<int>, i: int, j: int)
  returns (r: int)
  requires is_prefix_sum_for(a,c) ∧ ....
  ensures ...
  { ... }
```

As the signature above suggests, you will need to define a *predicate* that codifies the relation of being a prefix sum. The predicate should be defined as follows:

```
predicate is_prefix_sum_for (a: array<int>, c: array<int>)
  reads c, a
{ ... }
```

where the body of the predicate is a logical formula that characterizes the prefix sum array `c` in terms of the array `a` (and its contents in terms of `sum` of elements of `a`).

**Hint:** The body of method `queryFast` should be extremely simple – think of how to compute a sum over a range $[i, j[$ in terms of sums over $[0, i[$ and $[0, j[$. However, to establish the appropriate post-condition for method `queryFast` you will likely need to *call* upon a lemma (which you will have to state and ensure that Dafny can prove) that identifies the fact that the interval sum over $[i, j[$ is equal to the sum over $[i, k[$ plus the sum over $[k, j[$, for any appropriate $j$.

## 2 Functional Lists and Imperative Arrays

Now that you have suffered a bit with prefix sums over arrays, your task is now much simpler. We start with the standard definition of (functional-style) lists in Dafny:

```
datatype List<T> = Nil | Cons(head: T, tail: List<T>)
```

All we ask is that you implement and prove correct an imperative, non-recursive method:

```
method from_array<T>(a: array<T>) returns (l: List<T>)
requires ...
ensures ...
{ ... }
```

That takes any `a: array<T>` and produces a value of type `List<T>` containing exactly the elements of the given array. You should try to make your specification as precise as you can. The more precise specification you can prove, the better your grade!

**Hint:** You will need to define a notion of membership over lists in terms of functions or predicates. We suggest you define something along the lines of: `f`unction mem<T(==)> (x: T, l: List<T>) : bool ... which returns **true** if and only if the element $x$ is in list $l$. Note that you will not need to provide a specification for this function / predicate, but you will need to use it in the specification of `from_array`.

# 3 Grading

Your submission must have non-trivial postconditions in order to be graded. Your preconditions cannot be equivalent to false (i.e., it must be possible to invoke your methods).

If your submission is not fully checked by Dafny, your grade will be significantly penalized.

Exercises in Group 1 are with 65% of the full grade, the exercise in Group 2 is worth 35% of the full grade. For Group 2, your grade will be a function of how precise your *proved* specification is.