DCC831 Formal Methods
2022.2

Mini Project 2

Due: Monday, December 19, by 11:59pm

Download the accompanying files

`project2a.dfy` and `project2b.dfy`

and complete them as specified below. Type your name(s) as indicated in the files, and submit them on Moodle.

For teams, *the same team member should submit the files*, but make sure to write down the name of both team members in the source files.

This project has two parts which can be done independently. It will test your ability to write programs in the Dafny language, annotate them with specs, and check their correctness. Use Dafny to verify that your programs are correct with respect to their specification. The implementation effort requires only very basic knowledge of object-oriented programming. *You are strongly advised to do all the assigned readings on Dafny before attempting to do it.* You should also check the Dafny reference manual for more information whenever necessary.

Note that you will also be evaluated by correctly capturing in your specification the requirements stated in the document and accompanying files. Contracts which are too weak (i.e., do not enforce the requirements) will be penalized.

Part A

File `project2a.dfy` contains a prototype of an email client application based on the one we specified in Alloy earlier in the semester. The functionality meant to be provided by the prototype is fully implemented in the classes `MailApp` and `Mailbox` as specified by the English specification in comments in the classes, but it lacks any formal specification. Add one so as to fully capture the English specification. Express method contracts strictly in terms of the abstract state of each class. For simplicity, in `Mailbox` the concrete and the abstract state are the same—i.e., there are no ghost fields. In contrast, in `MailApp` the abstract state consists of the ghost field `userboxes` and four non-ghost fields: `inbox`, `drafts`, `trash`, and `sent`. Field `userboxes` is implemented in concrete with the aid of non-ghost field `userboxList`. Add code as needed to the body of `MailApp`'s methods to maintain the connection between abstract and concrete state.

As usual, also annotate the code with reads, modifies and decreases clauses as needed for Dafny to be able to prove the correctness of the implementation with respect to your specification.

The `MailApp` and `Mailbox` rely on a few other classes and auxiliary functions. Among these, the class `Message` has a full formal specification but no implementation for its methods. Provide one yourself according to the formal spec. The file also contains the generic function method `rem` that removes a given element `x` from a given sequence of elements of the same type as `x`. Provide an implementation for this method too, consistently with the provided formal specification.

Part B

File `project2b.dfy` contains an initial, partial definition of a generic Dafny class `Buffer<T(0)>` implementing a fixed-size buffer of elements of an arbitrary default-initializable type `T`. Data is written to the buffer, with method `put`, and read and removed from the buffer, with method `get`, in a FIFO fashion. Writes to a full buffer and reads from an empty buffer are not allowed.

As an abstract datatype, the buffer is just a sequence of data elements that can grow up to a predetermined length, the *capacity* of the buffer, with new elements added at the end and old elements removed from the front. The buffer and its capacity are modeled abstractly in `Buffer<T(0)>` respectively by the ghost variables `Contents`, a sequence of elements of type `T`, and `Capacity`, a natural number.

In concrete, you are to implement the buffer as a circular buffer using an array field `a` together with an integer field `front` that refers to the oldest element in the buffer, and an integer field `size` that stores the current number of elements in the buffer.

The buffer starts empty. Reads, each of which consumes the currently oldest element, cause `front` to advance. Writes are done in consecutive positions of the array `a`, starting from the first available position; they wrap around (that is, restart from position 0) once an element is written at the last position of the array (`a.Length-1`), as long as the buffer is not full.

The various methods of the class have a fully specified contract in terms of the abstract state. Do not modify those contracts. The `Valid` predicate, representing the class invariant, is defined only partially. Complete its definition by adding suitable concrete state invariants.

Implement `Buffer<T(0)>` as required above by filling in the definition of its various methods. Annotate the Dafny code with loop invariants and reads, modifies and decreases clauses as needed for Dafny to be able to prove the correctness of your implementation with respect to the specified contracts.

Hints

1. If you use Dafny within Visual Studio Code, open and work on one file at a time.
2. When working on a method it might be useful to temporarily comment any other unrelated methods.
3. Since Dafny checks requires, ensures, and invariant clauses incrementally, the order in which you write them sometimes matters, even if it should not from a logical point of view. So for example, write something like

```
invariant 0 <= i < a.Length;
invariant a[i] > 0;
```

instead of

```
invariant a[i] > 0;
invariant 0 <= i < a.Length;
```

to establish that `i` is within bounds before it is used in `a[i]`.

4. It might be useful to define predicates which capture the expected semantics of the methods you are writing contracts for and ensure that they are respected by the implementations.

Submission Instructions

You will be reviewed for:

- The clarity of your implementation and annotations. *Keep both short and readable.* Submissions with complicated, lengthy, redundant, or unused code or specs may be rejected.
- The correctness of your code with respect to original specification.
- The correctness of your annotations.

For each part, *your Dafny code should be free of syntax and typing errors.* You may get no credit for that part otherwise. Submission with verification errors or warnings will receive partial credit.