# Software Specification (ES) 2018/2019
# Instituto Superior Técnico
# 2$^{st}$ Project

**Deadline: December 12, 2018, 22:00**

## 1 Introduction

The goals of this project are to practice the development of verfied software systems, in particular using Dafny, Dafny's various language constructs, and some custom extensions to Dafny in order to implement some file-manipulating programs.

For this assignment, we provide the boilerplate code as well as samples files. They are available from the course unit's website.

Please answer the questions in this script in a **REPORT.md** file (use Markdown to write the readme file).

## 2 Background

Dafny can be extended by binding trusted Dafny interfaces to $C\#$ code. For example, by default, Dafny does not support any command-line arguments. However, we can extend it to do so as shown in the `Io.dfy` and `IoNative.cs` files provided. Note that the class `HostConstants` has methods to determine how many command-line arguments were provided, as well as to retrieve them. It also has a function that lets you talk about those arguments in specification contexts. Note that each one is marked as `{:extern}`, which means both that the interface is axiomatically trusted and that Dafny will expect us to provide a $C\#$ implementation of each executable method. This is exactly what `IoNative.cs` provides. Notice that the names used in IoNative.cs carefully line up with those chosen in `Io.dfy` (though this isn't necessary if you specify the names when providing the extern annotation). To connect the two, pass `IoNative.cs` as an extra command-line argument to Dafny when compiling.

To make use of the IO routines, you can use Dafny's include mechanism. In another file, just write `include "Io.dfy"` at the top-level of the file, and include `IoNative.cs` on your command line when invoking Dafny.

**Question 1:**

Note that in the `FileSystemState` and the `FileStream` classes, all of the functions say they read `this`. Why is this important?

**Question 2:**

Note that it isn't possible to create new `FileSystemState` objects. What would problems might arise if this were possible?

**Question 3:**

Semantically, what does it mean if you add preconditions (requires) to the `Main` method?

## 3 Challenge 1: File Copy Utility

Write a specification for `Main` that defines a copy utility. That is, the program expects two command-line arguments, source and destination, and copies source to destination, assuming destination doesn't already exist:

```
./cp SourceFile DestFile
```

Be sure to make the specification as strong and precise as you can, given the interface defined in `Io.dfy`.

Write an implementation and prove that it meets your specification. Put your solution in a file called `cp.dfy` and include it, along with any other files you depend on, in the `cp-basic` folder.

**Bonus Points:**

Extend `Io.dfy` and `IoNative.cs` to permit basic user input. Extended your specification so that when performaing a copy, if the file already exists, the user is asked if the file should be overwritten.

Extend your implementation to support this functionality, and prove that this is the only time when you will overwrite an existing file. Put your solution in a file called advanced-cp.dfy and include it, along with any other files you depend on, in the `cp-adv` folder.

## 4 Challenge 2: Verified Compression Algorithm

The goal of this assignment is to verify something larger and more ambitious than what what we have done before. Specifically, you will write a utility for losslessly compressing and decompressing files. Your program should run from the command line as:

```
./compression 1 SourceFile DestFile
```

to compress `SourceFile` into `DestFile`. Given a `0` instead of a `1`, it should decompress `SourceFile` into `DestFile`. In your file, you should prove that calling decompress on compress is the identity function. **Note, however, that you will not receive credit for implementing compress and decompress as the identity function!** Be sure to document any references you use for your compression algorithms.

We have included three example files for you to run your compression and decompression routines on. We will test your solution on similar but not identical files. If your verified compression routine produces a smaller file for at least one of the three, you will receive full credit. If you compress all three, you will receive extra credit. The verified implementation that compresses the most will receive additional extra credit (in the event of a tie, the solution that runs the fastest will win). You can also get partial credit if you at least write a functional specification for compress and decompress and prove that they are inverses, even if you don't manage to implement them successfully. You are encouraged to comment your code, so we can understand your design decisions.

Futhremore, extra credit will be given to those groups that write a REPORT.md , *a la* blogpost [1], explaining the algorithm and decisions. If appropriate we will publish the reports on Medium – let's go for creating impact!

## 5 Hand-in Instructions

The project is due on the **12th of December, 2018, 22:00**. You should follow the following steps to hand-in Project 2.

**Preparing submission:**

- Make sure your project is named `DafnyGXXP11819.zip`, where `XX` is the group number. Always use two digits, that is, Group 8's project should be named `DafnyG08P11819`.
- `DafnyGXXP11819.zip` is a zip file containing two the solutions as well as the README)

**Upload the file in Fenix** Upload the files `DafnyGXXP11819.zip` in Fenix prior to the respective deadline.

## 6 Project Evaluation

### 6.1 Evaluation components

In the evaluation of this project we will consider the following components:

1. Correct answers to the questions in the script: 0–3 points
2. Challenge 1: Correct development of the model and specification of the requirements: 0–5 points. (**pre-conditions; post-conditions**)

[1] E.g., `https://medium.com/@ahelwer/formal-verification-casually-explained-3fb4fef2e69a`

    - 2 point for the basic version
    - 3 points for the advanced version
3. Challenge 2: Correct development of the model and specification of the requirements: 0–8 points (**Compression rate; pre-conditions; post-conditions; loop invariants**)
    - Partial credit: 2 points
    - Full credit: 6 points
    - Extra credit: 2 point
4. Quality, completeness, and simplicity of the obtained solution: 0–4 points.

If any of the above items is only partially developed, the grade will be given accordingly.

### 6.2 Other Forms of Evaluation

It may be possible *a posteriori* to ask the students to present individually their work or to perform the specification of a problem similar to the one of the project. This decision is solely taken by the professors of ES. Also, students whose grade in the first test is lower than this project grade by more than 5 may be subject to an oral examination.

In both cases, the final grade for the project will be individual and the one obtained in these evaluations.

### 6.3 Fraud Detection and Plagiarism

The submission of the project presupposes the **commitment of honour** that the project was solely executed by the members of the group that are referenced in the files/documents submitted for evaluation. Failure to stand up to this commitment, i.e., the appropriation of work done by other groups, either voluntarily or involuntarily, will have as consequence the immediate failure of this year's ES course of all students involved (including those who facilitated the occurrence).

## 7 Useful Development Tips

### 7.1 General Tips

- Start early. Verification will take longer than you expect.
- Work in small, incremental steps. Don't try try to write all of the code at once and then go back and prove it correct.
- Using small, well-factored methods will likely help.
- Even though we're verifying everything, comments in the code are still encouraged! They help explain your high-level ideas to others, or even to future versions of you.

## 7.2 Tips to write better specifications

- Know your Boolean operators!
  - If the method must satisfy different post-conditions based on the input value, use an implication to describe those conditional behaviors

    $$ensures\ P ==> Q1$$

    $$ensures\ !P ==> Q2$$

    The condition P should be based on values before the execution (and therefore use $old$), otherwise these assertions might not mean what you think they do
  - Avoid writing $b == true$ and $b == false$ when testing Boolean values, prefer $b$ and $!b$ instead.
  - If a condition must be satisfied if and only if another condition is true (for example "the method returns true when the element has been added to the data structure") use an equivalence $<==>$ rather than multiple implications.
- Use the right data types
  - Like Haskell or ML or Python, Dafny has algebraic data types that can be used (among other things) for enumerations or tuples.
  - There are also built-in types for sets, multisets, sequences, maps. Use them to your advantage and don't limit yourself to arrays.
  - Choosing the right data type not only makes the specification much easier to read, but data types are in themselves a form of specification!
- Keep it small and simple
  - In the modifies clause, include only what is actually modified and nothing else. Avoid $modifiesthis$ as it means that any field of this object can be modified. Instead detail exactly the fields that are modified, and only those.
  - Keeping things simple is useful when coding, but even more important for specification. Your specification should be complete, but if it is complex and unreadable it is not likely to be useful (or correct).

## 7.3 Common Error Messages

- **call may violate context's modifies clause**

  This error can be caused by a method writing in a location that is not included in it "modifies" clause. However if you get it in your main program, it is likely because a method initialized a field without ensuring that it was a fresh location. The explanation is that the prover cannot know by default where the field points to, so the next time you modify it, you may affect any location in the memory.

  **The fix**: for any method or constructor that initializes a field $f$ with a call to $new$, make sure that the specification includes $ensuresfresh(f)$. With this indication the prover knows that the field refers to a memory location that is not shared with any other object.

- **assertion violation**

  This error could mean that there exists conditions under which an assertion (signaled by the command assert or ensures) is violated. But it could also mean that the prover simply does not have enough information to prove the assertion. When the prover encounters a call to a method (in the main program or inside another method), it is "blind" to the method body, and instead only uses the information given by the contract of the method, which also already be proven. This is allows the prover to more efficient and ensures a modular design.

  **The fix**: make the post-conditions of your methods stronger, so that they describe exactly what the method does.

- **index out of range**

  The prover checks that array accesses are only done within the bounds of the array, both in the program body and in the assertions. In order to do that it must have enough information about the indices and how they relate to the array length.

  **The fix**: make the pre-conditions of your methods stronger, in particular concerning the indices. If an index is modified, indicate its new value in the post-condition. In post-conditions, make sure that you do not confuse the value of a index $i$ before the method $old(i)$ and its value after, $i$.

- **array may be null**

  The prover also checks whether arrays have been initialized. This is usually straightforward but if your modifies clause is too generous, the prover may assume that a previously initialized array reference has been modified.

  **The fix**: make your modifies clause as precise as possible. Be particularly careful with modifies this, which means that any field of the object may be assumed to have been modified.

## 8 Final Remarks

All information regarding this project is available on the course's website, under the section *Project*. Supporting material such as links, manuals, and FAQs may be found under the same section.

In cases of doubt about the requirements, or where the specification of the problem is possibly incomplete, please contact the Professors of the course.

GOOD LUCK!