Lab 11

Due Wednesday, May 14 by 11:59pm

Handout 24
CSCI 334: Spring 2025

In this lab, you will complete your final project. To help you finish, I provide a final project checklist below. Be sure to pay attention to the minimum required amounts for writing where given (e.g., **2+ paragraphs**).

Two of the items on the checklist are new to you:

- writing a Semantics section, and
- implementing unit tests.

Both new items are described in detail below. You should be familiar with the other items on the checklist.

Turn-In Instructions

For this lab, you will continue to use your project repository. Be sure to follow the instructions for committing your work to the appropriate branch.

Turn in your work using your assigned `git` repository. The name of your repository will have the form `https://aslan.barowy.net/cs334-s25/cs334-project-<USERNAME1>-<USERNAME2>.git` For example, if your CS username is `abc1` and your partner's is `def2`, the repository would be `https://aslan.barowy.net/cs334-s25/cs334-project-abc1-def2.git`

Pair Programming Assignment

This is a pair programming lab. As with previous partner labs, you may work with a partner. However, for a pair programming assignment like this one, you may collaborate to produce a single solution. You do not need to submit a `collaborators.txt` file for this lab.

This assignment is due on Wednesday, May 14 by 11:59pm.

Reading

1. **(Required)** Read "Unit Testing in F#" from the course packet.
2. **(Required)** Read "Appendix B: Branching in `git`" from the course packet.

Problems

Q1. Set the Project Branch

Your work must be committed to a branch called `final`.

To create and switch to a `final` branch:

(a) Run `git checkout -b final`, which will create a new branch called `final`.

(b) Make your changes, then `git add` and `git commit` as appropriate to save your changes.

(c) To push the new branch to `aslan` for the first time, run `git push -u origin final`. We need to push differently than usual because the `final` branch you just created does not exist on the server. Subsequent calls to `git push` can be made as usual.

(d) Go to your repository and verify that your new `final` branch appears in the web interface.

After you have pushed the `final` branch to the server, if your partner wants to check out the same branch, they should first `git pull` and then run `git checkout final`.

Merge conflicts. Be aware that when any two developers work on the same branch, it is always possible that `git` will not be able to automatically "merge" the two sets of changes. This common occurrence is called a "merge conflict." When you see this, keep in mind that it is a safety feature that prevents you from accidentally overwriting another developer's changes. It is only frustrating when you don't know that conflicts are normal. You should expect merge conflicts to happen, and in a large enough project, they will happen with some frequency. The required reading walks you through handling them.

Q2. Semantics

Your final project specification should explain the semantics of the constructs in your program. *A language semantics explains how syntax is converted into an AST node (or nodes) and what that AST fragment means.* A good choice at this stage is to describe one of your language's primitives (e.g., data) and one of your language's combining forms (e.g., an operation). For example, you might build the following table.

Syntax	**Abstract Syntax**	**Prec./Assoc.**	**Meaning**
`<n>`	`Number of int`	n/a	n is a primitive. We represent integers using the 32-bit F# integer data type (`Int32`).
`<expr> + <expr>`	`PlusOp of Expr * Expr`	1/left	`PlusOp` evaluates two expressions, $e1$ and $e2$, adding their results, finally yielding an integer. Both e_1 and e_1 must each evaluate to `int`, otherwise the interpreter aborts the computation and alerts the user of the error.

Your semantics does not need to be formal, and it does not need to be in a table, but it should discuss the items shown the table above. You are required to document all of the parts of your language.

Your semantics should be added to the specification document that appears in your "`docs`" folder. You must use LaTeX for your specification.

Q3. .... Tests

This submission is required to have at least five tests.

(a) At least one test should be an "end-to-end" test that ensures that for a given program in your language you get a given output. An end-to-end test should invoke both your parser and your evaluator.

(b) At least one test should test one of your parser functions. Any parser is fine.

(c) At least one more should test one of your evaluation rules. Any evaluation rule is fine.

(d) For the remaining tests, you may choose any other component that you think would be helpful to you.

Setup. To implement tests, you will need to create a **solution** for your project. Creating a solution will require you to rearrange some of your files inside your `code` folder. Be sure to do this reorganization, as it is an important part of the final checklist. See the unit testing reading in the course packet, and the "Running tests" subsection below, for guidelines.

Rationale. From personal experience developing languages, tests are tremendous time-savers. It is always frustrating to discover that a newly-added feature breaks other functionality. Making that discovery well after you've added the feature—that's even worse. Having a good test suite will help you find problems early, and it will save you a lot of sweat and tears. If you want to have more than five tests, please go ahead and implement them.

I strongly recommend testing your parsers, which are pure functions and are therefore relatively easy to test. To test parsers, you will first need to `prepare` your input string, then pass it to one of your parser functions, then check for `Success` or `Failure` in your test.

Running tests. Your should be able to run your tests by running `$ dotnet test` from your `code` directory. Since tests run in the folder that your `sln` file resides, you will need to reorganize your project. Your solution should be in the `code` folder, your language implementation should be in the subfolder `code/lang`, and your tests should be in a folder called `code/tests`.

Q4. .... Final Project Checklist

(a) ________ You have created a 5-10 minute video presentation, and included this video with your implementation. Note: large video files cannot be added to `git`; please include a link to Google Drive/YouTube/etc. instead.

(b) ________ Your project has a (silly) name.

(c) ________ Your project has a specification.

(d) ________ Your parser is in a file called `Parser.fs`.

(e) ________ Your AST is in a file called `AST.fs`.

(f) ________ Your interpreter / evaluator is in a file called `Evaluator.fs`.

(g) ________ Your `main` function is in a file called `Program.fs`.

(h) ________ Your project compiles (this is **very** important).

(i) ________ Your project runs (this is **very** important).

(j) ________ Your language "does something." It prints out a computed result, it generates a file, etc.

(k) ________ **You are sure to tell the user** (me) what the expected result should be!

(l) ________ Your implementation has a test suite and it runs.

(m) ________ There is at least one test written for the parser.

(n) ________ There is at least one test written for the interpreter.

(o) ________ Your project can be run at the project level by calling `dotnet run <input>` or at the solution level by calling `dotnet run --project <whatever.fsproj> <input>`.

(p) ________ Your specification document has a title.

(q) ________ Your name and your partner's name is written at the top of the spec.

(r) ________ Your specification has an **Introduction** section consisting of **2+** paragraphs.

(s) ________ Your specification has a **Design Principles** section consisting of **1+** paragraphs.

(t) ________ Your specification has an **Examples** section consisting of **3+** example programs.

(u) ________ Each of your examples is provided (and ideally, each example is in a separate file) so that an interested third party (me or one of your classmates) can run them. Instructions to run the examples is provided.

(v) ________ Your specification has a **Language Concepts** section consisting of **1+** paragraphs.

(w) _________ Your specification has a **Formal Syntax** section consisting of **as much BNF is needed** to completely describe your language's syntax.

(x) _________ Your specification has a **Semantics** section consisting of **one short description per language element** (where an element is normally an AST node), completely describing your language.

(y) _________ Your specification provides enough detail that an interested third-party (like me or one of your classmates) can write a new program in your language.

(z) _________ If your parser is "generous" and accepts programs that actually do not make sense, make sure that the program detects these cases and shuts down cleanly (i.e., the user does not see an exception).

(α) _________ You committed your specification, in a folder called `docs`, to a branch called `final`.

(β) _________ You committed your implementation, in a folder called `code`, to a branch called `final`.

(γ) _________ Your specification has a **Remaining Work** section that describes further enhancements, if any, you would like to see. If your prior draft had features that you did not get to by the final submission, briefly explain why you were not able to implement them.

(δ) _________ If you would like me to transfer ownership of your repository to you, please provide the name of a Github user or organization that will take over ownership. Put this username in a file called `TRANSFER.txt`.

(ϵ) _________ If you intend to make your repository public, put a `LICENSE.txt` copyright statement in your repository, at the root level, so that people know under what conditions you plan to let them use your code. For example, provide a copy of the GNU Public License, or BSD License, etc.

(ζ) _________ Is it OK for me to share your project with future CSCI 334 students? Students often tell me that it is helpful to see what others did. If so, please add a file called `SHARING.txt` to your repository. If you have any conditions on sharing, please put your notes in the file, otherwise it's OK to leave the file blank. If you do not want to share, do not create the file.