Build your own language
LINGI2132

March 15, 2021

1 Context

Through the year, you will build your very own language by group of 2 (we won't allow smaller or larger groups). We do not impose limits on what you can do, but we require that you support **at least** the following set of features

- Literals: Integer (at least 32-bits signed precision, `1`, `-999`, ...), Strings (`"Hello"`), boolean
- Comments (e.g, `//`, `#`)
- Variable definitions (e.g., `var` or `let`)
- Simple arithmetic (at least `+`, `-`, `*`, `/`, `%`) with priority of operations (i.e., `1 + 2 * 3` is equivalent to `1 + (2 * 3)`).
- Conditions (e.g., `if`). You should be able to compare: two integers; two variables; one integer and one variable; two boolean values and compositions of these (at least the 'and', 'or' and 'not' operators)
- Loops (e.g., `while`)
- Function definition (e.g., `function`, `def`, `defun`, etc): they must be able to take parameters and return a value.
- Array and map access (e.g., `array[index]`, `map[key]`) as well as function call (e.g., `foo(1, 2)`)
- Printing to stdout (e.g., `System.out.println`, `print`) (will typically be a function)
- a way to pass string parameters to the program: it could be a `main` method or could be implicit `$1`, `$2`, ... parameters (at least 9 parameters must be supported)
- A way to parse strings representing numbers to integers

Note that you might have a brillant idea that contradicts these requirements. In that case you can send us an email (include the whole teaching team) and we will discuss together to see if we can adapt these requirements for your use case.

The project will be divided in four parts:

1. The parsing with the AST generation
2. The semantic analysis
3. The interpretation of the code/ code generation
4. The final submission with extra features (more info in Section 5)

Here is an overall calendar for the whole project, note that it might change a bit with how the situation evolve:

Part	Start	end
Parsing	S3	Friday S6
Semantic analysis	Monday S7	Friday S9
Interpeter	Friday S9	Friday S12
Final submission		Friday S13

Each part will have its own deadline and will be graded independently. We expect you, **at each part**, to do **at least all the required features**. For instance, in the first project we expect you to generate the AST for all the features listed above, but we encourage you to do more if you have time. On the other hand, if you did more features in the first part, we **do expect that you first handle the basic features for the second part**. We will not grade the additional features if you do not give us the minimum set of expected features.

You might add/change features as time pass (it is normal, your language will evolve!), do not worry **we will not decrease retroactively your grade**. Basically it means two things: i) You can still add functionnalities (new grammar rule) even if we are at the last part of the project ii) If you could not finish all the requirements for a given part, finish them for the next part to avoid losing more points.

We encourage you to work incrementally, start with small and easy things, then work up the more difficult/complex part of your language. It does not make sense to start looking how you will implement conditions if you do not have booleans.

2 The design of your language

The first task we ask you to do is to think about your language (this part is obviously not graded, but important). Here is a list (**non-exhaustive**) of things you can decide beforehand (remember that you can still change during the course of the semester if you want!)

- Is your language statically or dynamically typed?
- How do you handle errors (e.g., array out of bounds, used of uninitialized variable, division by zero)?
- What are the default values for your types? Do you allow to separate the variable initialization and declaration?
- What nice **extra** features do you want to have (list comprehension, streams, reactive programming)?

It is nice to have advanced features, but try to keep your to-do list manageable and see at the end if you have time to add more things.

3 Part 1. Grammar and Parsing

During the first assignment, you will have two main tasks:

- Implement a parser for your grammar with Autumn
- Write a comprehensive list of automated unit tests for your parser

Disclaimer: We expect you to provide a comprehensive set of unit test for your parser. If you do not, we will assume that your code does not work.

Your parser will be implemeted in Java 8 or higher (the grading environment will be Java 15 but the version are retro-compatible for the most part, so do not worry) with Autumn library. You should both be able to correctly parse program in your language and produce its Abstract Syntax Tree. For details on the submission of your parser, see Section 6.

Automated testing your parser As a rule of thumb, every rule should be tested **in isolation**. In addition to that, we expect you to provide programs (in your language) for these problems:

- The FizzBuzz function
- Find the first N (argument of the program) prime numbers
- Print the first N (argument of the program) element of the Fibonacci sequence (use a recursive function)
- A program that takes a serie of Strings as parameters and print them on `stdout` omitting duplicates (use a map)
- A program that takes N integers (arguments of the programs) and print them in increasing order

These small program will be used throughout the semester to test your language. An example of implementation is given in Java. You can also add programs that show exceptional features of your language!

4 Part 2. Semantic analysis

During the second assignment, you will have two main tasks:

- Implement a semantic analysis for the AST produced by your parser with Uranium
- Write a comprehensive list of automated unit tests

Disclaimer: As for parser, we expect you to provide a comprehensive set of unit test. If you do not, we will assume that your code does not work.

Your analyzer will be implemeted in Java 8 or higher (the grading environment will be Java 15 but the version are retro-compatible for the most part, so do not worry) with Uranium library. You should both be able to correctly analyze a program in your language and validate it. For details on the submission of your semantic analysis, see Section 6.

Automated testing your parser As a rule of thumb, every rule should be tested **in isolation whenever possible**. It is sometimes impossible to test every rule in isolation for semantic analysis. For instance, checking the correct use a variable depends on its declaration.

You should still have tests that run the semantic analysis on the programs requested in the previous section: FizzBuzz,... As always, you can also add programs that show exceptional features of your language!

5 Grading

The minimum requirements presented before account for roughly 2/3 of the total points of the project and each of the three parts account for the same part in these 2/3. To have more than that you can

- Add some extra features to your language. Note that these need not be "user-visible" features.
- Add some interesting analysis about the behaviour of your language and its implementation.

These extra points will be evaluated only at the end of the project (see next section for the overall deadlines). Remember that we only evaluate these if you correctly implemented the required features.

6 Deliverables

We ask you to submit, before **18:00 the Friday 12th March 2021 (S6)**, on **INGInious**:

- An archive with your source code and your tests. Include in your archive only the minimum required files (The gradle files, the sources and your tests (`.java` files). **Please do not include your hidden files, we do not want to have them on our computer when we download your submissions!**).

- A report of maximum 2 pages, in pdf format, explaining the features of your language

We ask you to submit, before **18:00 the Friday 2nd April 2021 (S9)**, on **INGInious**:

- An archive with your source code and your tests. Include in your archive only the minimum required files (The gradle files, the sources and your tests (`.java` files). **Please do not include your hidden files, we do not want to have them on our computer when we download your submissions!**).
- A report of maximum 2 pages, in pdf format, explaining the features of your language and how it impacts the semantic analysis

Here is an overall calendar for the whole project, note that it might change a bit with how the situation evolve:

Part	Start	end
Parsing	S3	Friday S6
Semantic analysis	Monday S7	Friday S9
Interpeter	Friday S9	Friday S12
Final submission		Friday S13