CMPU 331 · Compilers · Fall 2019

Project - Lexer

Due Tuesday, September 24, 2019 by 11:59 p.m

1 Overview

The four project milestones will direct you to design and build a compiler for the DL language, which is a simple subset of the C language. Each assignment will cover one component of the compiler: lexical analysis, parsing, semantic analysis, and code generation. Each assignment will ultimately result in a working compiler phase which can interface with other phases. You will be implementing the projects in Python.

For this assignment, you will write a lexical analyzer (”lexer”) using a lexical analyzer generator called ”Sly Lex-Yacc” (or ”SLY”). You will describe the set of tokens for DL in an appropriate input format, and SLY will provide the actual code for recognizing tokens in DL programs.

You may work either individually or in pairs for this project.

2 Getting Started

You will submit your code through GitHub Classroom, so the first step is to clone a copy of the skeleton files from GitHub. In Moodle, under the topic ”Lexical Analysis”, you'll find a link to ”Lexer on GitHub”. Follow that link, and then click the ”Accept this assignment” button to create your own private copy of the skeleton files. Your copy will be a GitHub repository located at `https://github.com/vassar-cs331-2019b/lexer-yourusername` (substitute your GitHub username). From that page, you can copy the URL for cloning by clicking the green button ”Clone or download”, and then paste the URL onto the command-line as an argument to the `git clone` command.

```
$ git clone https://github.com/vassar-cs331-2019b/lexer-yourusername.git
```

The clone command will create a directory named `lexer-yourusername` on your machine, and in that directory you'll find the following files:

- `LICENSE` - This file contains the open source license for the code. Below the copyright line with my name on it, add another copyright line with your name (you are now the proud owner of your very own open source project fork).
- `setup.py` - This file contains metadata about the code. Replace my name and email with yours.
- `runtests.py` - This script is a simple test runner. You can call it with `python3 runtests.py` to run all the tests.
- `lexer_prompt.py` - This script is an interactive prompt for the lexer, which you might find useful while you're developing it. You can start the prompt by calling `python3 lexer_prompt.py`. For each line of DL example code you type, it will show you what tokens it found, or an error for invalid tokens. You can exit the prompt by typing `Control+D`.
- `sly/lex.py` - This file is a library you will use in your lexer. You won't make any changes to this file, it's only included in the skeleton so you don't need to install it.

- `dl/lexer.py` - This file contains a skeleton for a lexical description of DL. There are comments indicating where you need to fill in code, but you are welcome to make any modifications to the skeleton. The skeleton is functional, and will pass two tests, but it doesn't do much.
- `tests/test_lex.py` - This file contains tests for the lexer, including at least one test for each type of token, and one test of a longer code example. The provided tests are not exaustive, you may want to add more as you work. You won't be graded on the tests you add, but testing more thoroughly can increase your confidence that the lexer is working as it should. I'll incorporate the best student-added tests for the lexer phase into the parser phase, so everyone has the benefit of them.
- `tests/simple.dl` - This file is a code example in the DL language, which is used by one of the tests.

The CS lab workstations have all the software you need installed already. If you want to work on your own laptop, you might need to install git and Python 3, if you don't have them installed already.

PyCharm (`https://www.jetbrains.com/pycharm/`) is a good IDE for working with Python, and is already installed on the CS lab workstations. But, you can use any IDE or text editor you prefer.

3 Lexer Implementation

In this assignment, you are expected to write SLY lexer rules that match on the appropriate regular expressions, defining valid tokens in DL and performing the appropriate actions, such as returning a token of the correct type, recording the value of a lexeme, or reporting an error when an error is encountered. We will review SLY and Python in class, but you might also find it helpful to refer to these sources as you work:

- SLY documentation on writing a lexer: `https://sly.readthedocs.io/en/latest/sly.html`
- A full working example of a SLY parser (for now, you only need to pay attention to the lexer part): `https://github.com/dabeaz/sly/blob/master/example/calc/calc.py`
- Python documentation on regular expression syntax, for a list of special characters used in pattern matching: `https://docs.python.org/3/library/re.html#regular-expression-syntax`

3.1 Tokens

The following table provides a complete list of tokens and lexemes for DL, where the lexeme is either a literal string or an informal description of the pattern to match.

Token	**Lexeme**
`INTCONSTANT`	one or more digits
`IDENTIFIER`	a lowercase letter followed by zero or more lowercase letters or digits
`INT`	`int`
`IF`	`if`
`ELSE`	`else`
`PRINT`	`print`
`READ`	`read`
`RETURN`	`return`
`WHILE`	`while`
`SEMICOLON`	`;`

COMMA	,
OPENPAREN	(
CLOSEPAREN	)
OPENSQUARE	[
CLOSESQUARE	]
OPENCURLY	{
CLOSECURLY	}
PLUSOP	+
MINUSOP	-
MULTIPLYOP	*
DIVIDEOP	/
LEOP	<=
LTOP	<
GEOP	>=
GTOP	>
EQOP	==
NEOP	!=
ASSIGNOP	=

3.2 Skipped Tests

In the skeleton, many of the tests are marked as `skip`, because we don't expect these tests to pass yet. Marking the tests as skipped makes them display a clean message when you run `runtests.py`, like this:

```
... skipped 'Constant tokens not implemented yet'
```

When you start working on the tokens related to a particular test, first delete the line that marks that test to be skipped, from the test file `tests/test_lex.py`. The line will look like this:

```
@unittest.skip("Constant tokens not implemented yet")
```

After you delete the skip marker, the test will fail, and display an ugly error message. The error message will give you some clues about what's missing from your lexer. In the beginning it will probably complain most often about "Illegal character", or keywords being matched as the `IDENTIFIER` token. Once you've implemented all the tokens needed to pass a particular test, the status report at the end of the test run will change from something like:

```
FAILED (errors=1, skipped=4)
```

To something like:

```
OK (skipped=4)
```

Passing tests is how you know you've implemented your lexer correctly. When all the tests are passing with no skips, you're done.

4 Submitting

To submit your work, `git add` any files you changed, then `git commit` them with a sensible commit message about what you changed, and finally `git push` the changes back to GitHub.

```
    $ git add dl/lexer.py
    $ git commit -m "Added 3 more tokens"
    $ git push origin master
```

It’s a good idea to commit your work frequently as you go, so you have a backup if anything goes wrong. Only the last version you submit before the deadline will be graded.