CSE110A: Fundamentals of Compiler Design
Homework 3: Syntactic Analysis

Preliminaries

1. This assignment requires python3. This will be available on the provided docker, but you can also develop it locally. Please make sure it runs correctly on Gradescope. You can check if your output is incorrect Gradescope based on the auto-grading tests the HW3. (there are more instructions at the end of the document).
2. This homework contains 2 parts. Part 1 is worth 70 points; part 2 is worth 30 points.
3. If you need help, please visit office or mentoring hours. You can also ask questions on Piazza.

   You are allowed to ask your classmates questions about frameworks and languages (e.g. Docker and Python). You are not allowed to discuss technical homework solution details with them.
4. Your submission will consist of two parts: programming and explanation. It will be submitted through Gradescope. The programming part will be automatically graded, the explanation and textually filled out outcome.txt files will be manually graded afterwards.
5. It is highly recommended that you read chapter 3 up to 3.3 in the EAC textbook for this assignment to supplement the lectures

Parsing C-simple

The material for homework 2 will be critical for this assignment. You will use your grammar as well as your first+ sets. If you do not want to use your own output from homework 2, you can use the reference file: grammar-ref.txt, which will be available to you after homework 2 ends in a coupled days.

1 A recursive descent parser for C-simple

For this part of the assignment, you will write a recursive descent parser for C-simple using the grammar and First+ sets as a guide.

The skeleton for this part of the code can be found in the git repo. You will find a python file for the scanner (scanner.py), and a python file for the parser (parser.py). The driver is in main.py.

You can run the main.py similar to how you ran the scanners in homework 1. That is, you can run:

```
python3 main.py test_file_name
```

where test_file_name is the file you want to parse.

A successful parse will simply exit. An unsuccessful parse should throw an exception (which you are in charge of implementing).

1.1 Tokens and scanner

To begin this section, write the tokens you need for C-simple in scanner.py. You can use some of the token definitions from homework 1 or homework 2 if you would like. In addition, please implement a token action for keeping track of the line number as a scanner member function.

Currently, the token() method of the scanner object is left unimplemented. You should copy in one of the scanner implementations from Homework 1. I recommend using your SOS scanner, as it has decent performance, and allows tokens to be reasoned about independently. However, please keep the class name as Scanner and make sure to implement line number tracking as a token action on the IGNORE token.

Inside the tests directory are a few test cases. Please use them to verify your code. It can be executed as follows:

```
python3 main.py tests/test1.txt
```

1.2 Parser

The file parser.py contains a parser class with a constructor and one API function: parse.

It is up to you to implement the rest of the recursive descent parser.

The constructor contains an argument use_symbol_table; you will use that in part 2 of the assignment. Please do not worry about it for this part.

Each of the non-terminals from the predictive grammar should have a function inside the parser class.

You should follow the recipe for creating a recursive descent parser from the class lectures and from the book.

1.3 Errors

If you encounter an error, you should pass the following information to the parser exception: the line number (gotten from the scanner), the current lexeme that the parser is trying to match, and a list of tokens that would have been valid.

1.4 Tests

Inside the tests/ directory, please write 3-4 test cases to test your parser on. Inside tests/outcomes.txt, please write the expected outcome for each test: that is, if it should pass or fail, and on what line number it should fail.

1.5 What to submit

You will be graded on a completed parser.py and scanner.py files. You will be graded on the following criteria:

• Correctness: does your parser accurately parse the C-simple language? Are errors reported correctly? • Conceptual: does your parser implement your production rules from part 1 as a recursive descent parser? • Testing: did you design and run some interesting test cases?

2 Symbol Tables

In this part of the homework, you will add a symbol table to your parser. If the parser constructor is passed True for the use_symbol_table argument, then the parser should use a symbol table as discussed in class.

This can be enabled by passing the -s flag to the main.py driver

`Inside of parser.py, please notice the class skeleton for SymbolTable.`

You will need to implement the provided API.

Similarly, to check this part, we can use tests present inside the tests/ directory. Please use them to verify your code. It can be executed as follows:

```
python3 main.py tests/test1.txt
```

2.1 Checking IDs

Each time an ID is used in an expression, the symbol table should be checked to make sure the ID has been declared. If it has not been declared, then the parser should raise a SymbolTableException with the name of the ID that has not been declared along with a line number (from the scanner).

You should be able to add calls and checks to the symbol table inside the recursive descent parser completed in part 1. For example, IDs should be added to the symbol table after a declaration statement. Only perform the symbol table checks if the parser was constructed with use_symbol_table as True.

2.2 Scoping

We will follow the same scoping rules as C as applied to C-simple. A new scope is started after the opening brace in a block statement. The scope is popped immediately before the closing brace in the block statement.

2.3 Tests

Inside the tests/ directory, please write 3-4 test cases to test your new symbol table parser on.

Inside tests/outcomes.txt, please write the expected outcome for each test: that is, if it should pass or fail, and on what line number it should fail.

2.4 What to submit

You need to modify tests/outcome.txt, grammar.txt, parser.py, and scanner.py. Please zip your entire repository and submit the zipped file to Gradescope. As long as you submit one zipped file, the file name does not matter. (Please don't submit more than one)

Your grade will be based on 4 criteria:

• Correctness: Does your symbol table catch errors as it is supposed to? Does it implement the scoping rules correctly? • Conceptual: Is your symbol table implemented efficiently as a stack of hash tables (i.e. dictionaries)? Does your lookup traverse the stack appropriately? • Testing: did you design and run some interesting test cases? • Explanation: Is your code documented and readable

Submitting to Gradescope

Signing up for the GitHub Classroom assignment automatically creates a repository for you under the ucsc-cse110a organization.

This repository is for you to store the changes to your assignment, and will be where you submit your solution.

Do not rename the homework files.

Do not modify anything inside the .github/ folder.

The only files that should need editing for this assignment is parser.py and scanner.py. Furthermore, add the tests and outcomes.txt for your own testing and fill out answers to the questions in the Explanation.md part of Homework 3 in Gradescope.

Submit the python files to the Gradescope programming assignment. Submit the data from .txt and .md files to Gradescope as well, also submit the file doNotChangeMe.md5 unchanged.

Explanation assignment in Gradescope.

The results of the autograding passes can be seen in Gradescope. We reserve the right to run more tests than the ones provided to you.

As a reminder, it is advisable not to wait until the last minute to submit to Gradescope.