CS420 – Homework 4: Parser for Eck

Note: You will need to be running Python 3.8 or higher for this assignment

The task for this homework is to write a backtrack-free recursive descent parser for the Eck programming language. Your parser should determine if the input source file is a valid sentence in the Eck grammar, and, if so, return an intermediate representation of the source file using the abstract syntax tree classes provided in the *parserIR.py* file. If the source file is not valid, you should raise a `ParserError` with a message indicating the cause of the error and the line where the error(s) likely occurred.

If you wish, you may work on this homework with one other student from the class; be sure to list both students' names in all deliverables. As part of this assignment, you are specifically forbidden from:

- Using modules other than those already imported in the supplied files.
- Using a tool that automatically generates any part of your parser.

Supporting Materials

The assignment on Moodle includes a zip file containing the following files:

- *Eck Language Description.pdf*: A description of the syntax and semantics of the Eck language.
- *Eck Backtrack Free Grammar.pdf*: A BNF description of the Eck language after it has been transformed to eliminate left recursion and left-factored to make it backtrack free. This is the version of the language BNF your parser should implement.
- *lexeme.py*: Same as in Homework 1.
- *scanner.py*: Professor Hilton's implementation of the Eck lexical scanner. If your scanner did not correctly scan all of the test files in Homework 1, you should use this implementation. (You are also welcome to use this implementation if your scanner did pass all tests in Homework 1.)
- *parserIR.py*: A module containing a large number of classes for creating the abstract syntax tree structure that forms the parser's intermediate representation. The classes in this module correspond closely to the rules in the BNF grammar for Eck. The classes all begin with the prefix "PIR_", which stands for "Parser Intermediate Representation". You must use these classes to create your abstract syntax tree; do not roll your own. Do not modify the contents of this file!
- *eckTypes.py*: A module containing a number of enumerated type classes. You will need to use these enumerated types when you construct the parser's intermediate representation; look at the documentation for various classes in *parserIR.py* to see when you need to pass an enumerated constant to object creation functions. Do not modify the contents of this file!
- *pirPrinter.py*: A module containing a class for printing a PIR abstract syntax tree for testing and debugging your parser. Do not modify the contents of this file!
- *eckParser.py*: A module template for your parser that contains three things of importance:
    - The complete class definition for `ParserError`, which is a custom error class you should use for raising parsing errors.

    - An incomplete definition of the `Parser` class, containing several functions that facilitate the parsing task. You need to write methods that implement the BNF rules in the Eck grammar. You can include as many other methods and class definitions as you see fit to accomplish the task.
    - In the testing section of the module (i.e., the code after the `if __name__ == "__main__"` statement near the bottom of the file) is a script for testing your parser on the set of Eck files described below. I will be using this script to test your software.
- *ParserTests*: A folder containing ".eck" files for testing your scanner. The correct results that should be produced by the testing script are found in the corresponding ".answer" files. Note that not all of the ".eck" files are meaningful examples of the Eck programming language; they were written specifically to provide a robust test of the parser. Three groups of files are provided to aid in debugging your program at various stages of development:
    - There are a lot of BNF clauses dealing with expressions, so I recommend you deal with expressions only after you get the other parts of the language parsing correctly. The folder *"ParserTests/ExpressionlessSquare*" contains versions of the Square program that do not contain expressions (simple variables are used instead of expressions).
    - The file "*ParserTests/ExpressionTests.eck*" contains a thorough test of the binary and unary operators.
    - The remaining files in the "*ParserTests*" and "*ParserTests/Square*" folders test the full parser.

Deliverables

You need to flesh out the implementation of the `Parser` class. The testing code at the bottom of *eckParser.py* calls your parser and expects a `PIR_Class` object to be returned. The `PIR_Class` object is written to a ".parse" file. You should look at the provided ".answer" files to see what answer is expected. I will run this code to determine if your implementation produces the correct results. The source file line numbers in the .answer files do not need to match the numbers in your .parse file, but they should be close. I am not testing how your parser behaves for invalid source files.

You should upload to Moodle your *eckParser.py* module and any other modules you may have written as part of this homework. You do not need to upload the other modules provided with the assignment.

Grading Criteria

5%	In a comment at the top of each file, you include your name (and your partner's, if any), the course number, the assignment number, and the Eckerd honor pledge.
10%	Each of your modules, classes, methods, and functions are appropriately documented.
10%	Meaningful names are used for each of your modules, classes, methods, functions, and variables.
10%	Your modules are clean, not containing dead code, debugging print statements, or other cruft.
20%	All provided test files are correctly parsed.
45%	Quality and correctness of software design.