CMPSC461 Spring-2024
Programming Language Concepts
Instructor: Dr. Suman Saha

Project #1: Building a Parser

Introduction

This project provides hands-on experience with lexical analysis and abstract syntax tree (AST) construction, fundamental concepts in compiler construction and programming languages.

Instructions

1. You must submit file name as `project1_parser.py`. Any other file name will not be accepted by the GradeScope's autograder. Please strictly adhere to this format.
2. You must implement `Lexer`, `Parser` classes, and implement the `parse` function in the `Parser` class along with related functions.
3. Please generate your AST in the canonical form; refer to the example given in `test_utility.py`. Failure to comply will result in rejection of your submission by GradeScope.
4. You are advised to use git for maintaining your code changes throughout Project 1 and 2. There are many online tutorials available for git.
5. Please refer to `grammar.txt`, `example.txt`, and `test_utility.py` before beginning the project to understand the expected output format and the test cases.
6. If you get stuck, please reach out early. Do not wait until the last moment.

Grammar

Please refer to the `grammar.txt` in the project folder.

Example

Refer to `example.txt` for an example. A statement in the language `x = 5 + 3` matches the following Grammar rule:

1. `expression -> variable '=' arithmetic_expression.`
2. `arithmetic_expression -> term (('+' | '-') term) *`
3. `term -> factor (('*' | '/') factor) *`
4. `factor -> number`

Implementation

The file `project1_parser.py` contains lexer and parser classes and a few functions as guidelines for structuring your project. We highly encourage you to implement all those functions. If you decide to come up with your own implementation, please ensure you follow the below restrictions:

1. You must implement the `parse` function declared in the parser class in `project1_parser.py`. The checker will invoke the `parse` function to retrieve the AST representation of the program code to verify correctness.

2. Your output must match the pre-order traversal of the AST for each statement in the programming language. Please review the comment in `test_utility.py` before writing any code.

Grading Criteria

In addition to the 7 test cases given in `test_utility.py`, we have 3 more hidden test cases in the autograder script. The total points you can get in this project is 100 if you pass all the 10 test cases. Please note that all the test cases are not equally weighted.