Project Parts 3 & 4 – Parser & Static Semantics Analyzer
(50 points = 5% of course grade)

Introduction

The second delivery for the project contains two parts:

1. The Parser (35 points = 3.5% of course grade)
2. The Static Semantics Analyzer (20 points = 2.5% of course grade)

PART 3. PARSER

The Parser uses the grammar you defined in Part 1 to determine if the input token stream is syntactically correct; if it is, it generates a Concrete Syntax Tree (CST). This will be an opportunity to also review and possibly revise your grammar (or your language) if it turns out that the Parser cannot parse the programs you wrote.

You have a choice of which parser to use:

1. A Recursive Descent implementation of a LL(1) parser. We have seen a small example in class. This parser works by following closely the grammar. Starting from the start symbol of the grammar, it predicts based on the one token of input that it has access to, which RHS of a production it needs to use to parse the LHS nonterminal. Therefore, the first step in this part of the project is to double check that the grammar you have written, in addition step accurately describing WWL, is suitable for an LL(1) parser. Specifically:
    - It must be free of left recursion.
    - Each production must have RHSs that are pairwise disjoint on the first terminal.

    And, of course, it must be unambiguous.
2. A LR(1) or LALR(1) parser. For this, you will want to use an existing parser generator (for a wide choice check https://en.wikipedia.org/wiki/Comparison_of_parser_generators).

DELIVERABLES

1) A description of the parser you used, why you chose it, and how it works.
2) The parser code and information about what is needed to run it.
3) Examples of programs that test all the productions. Use a table that shows how each production (identified by the non-terminal being defined) and RHS, identified by a number (1st, 2nd, etc.) is tested by specifying which line or set of lines (numbered) in which program (identified by number or file name).
4) For each program, a readable view of the Concrete Syntax Tree produced (written to a file). Keep the test program and CST file names consistent.

PART 4. STATIC SEMANTICS ANALYZER

The Static Semantic Analyzer in this project is probably quite simple. The language you develop is unlikely to have a complex type system, so most of the work that will be done consists of:

1. Verifying that if an identifier is used it has been declared and defined in the right scope (if your language uses functions) by consulting information in the Symbol Table. This table may need to become a little fancier as a result.
2. Simplifying the CST into an AST to get ready for generation. You will need to design the AST structures corresponding to your syntactic structures and write an algorithm to transform the CST to the AST.

You can implement this part of the project as you see fit.

DELIVERABLES

1) Static Semantics Analyzer code
2) A description of how you implemented the tree decoration (if relevant) and defined-if-used check and Symbol Table use.
3) A description of the algorithm you used for converting from CST to AST.
4) For each program and CST, a readable view of the corresponding AST. Keep the AST, CST, and test program file names consistent.

CONNECTING THE PARTS

This topic was already discussed in Part 2 but reiterated here. You have a choice of

1. Writing the Lexer output (the token stream) to standard output, so it can be piped into the next stage with '>', or
2. Redirecting Lexer output to a file from which the Parser will read, or
3. Call the Lexer from the Parser by either:
    a. calling the Lexer directly
    b. having the Parser call a "Lexer" function that just reads from the file or stream of tokens where a separate and previous Lexer process has placed the token

How will you connect the Static Semantics Analyzer to the Parser? Probably operating directly on the CSTs in memory will be simplest.

It is okay to use different programming languages for different parts of your project, but you must find a way to make the communicate, whether through a pipe or a file.

DELIVERABLES

A brief description of how the different parts of your system communicate (Lexer-Parser-Static Semantics Analyzer).

Grading/Rubric

COMPONENT	POINTS
Parser	**34**
Parser Description and Code*	14
Examples and testing table	10
CSTs for each test program	10
Connecting the Parts	**1**
Static Semantics Analyzer	**15**
Analyzer code	5
Description of decoration/check	2.5
Description of CST-AST conversion	2.5
ASTs for each test program	5
TOTAL	**50**

* Point distribution depends on type of parser used