METROPOLITAN STATE UNIVERSITY OF DENVER
Department of Mathematical and Computer Sciences

CS 3210 - Principles of Programming Languages (Spring 2020)

Programming Assignment 01

Deadline: March 8th (11:59pm)

1. Introduction

The goal of this assignment is to have you write a lexical and syntax analyzer for a hypothetical programming language (roughly based on C). The input of your parser is a source code written in the programming language's grammar. If the source code is syntactically correct your parser should display the corresponding parse tree. An appropriate error message should be provided otherwise.

The expectation for this programming assignment is that you will be coding a complete parser from scratch. You can choose between the two approaches discussed in class: top-down or bottom-up. I will NOT grade solutions based on any parser generator like YACC or JavaCC, for example.

2. Grammar

Below is the grammar for the CLite programming language specified using EBNF notation. Special words and symbols are highlighted for easy identification (better seen in color). Assume that this PL is NOT case-sensitive.

```
<program>       → int main ( ) { <declaration>+ <statement>+ }
<declaration>   → <type> <identifier> [ [ <int_literal> ] ] { , <identifier> [ [
<int_literal> ] ] } ;
<statement>     → <assignment> | <if> | <while> | { <statement>+ }
<assignment>    → <identifier> [ [ <expression> ] ] = <expression> ;
<if>            → if ( <expression> ) <statement> [ else <statement> ]
<while>         → while ( <expression> ) <statement>
<expression>    → <conjunction> { || <conjunction> }
<conjunction>   → <equality> { && <equality> }
<equality>      → <relation> [ <eq_neq_op> <relation> ]
<eq_neq_op>     → == | !=
<relation>      → <addition> [ <rel_op> <addition> ]
<rel_op>        → < | <= | > | >=
<addition>      → <term> { <add_sub_op> <term> }
<add_sub_op>    → + | -
<term>          → <factor> { <mul_div_op> <factor> }
<mul_div_op>    → * | /
<factor>        → <identifier> [ [ <expression> ] ] | <literal> | ( <expression> )
<type>          → int | bool | float | char
<identifier>    → <letter> { <letter> | <digit> }
<letter>        → a | b | … | z | A | B | … | Z
<digit>         → 0 | 1 | … | 9
<literal>       → <int_literal> | <bool_literal> | <float_literal> | <char_literal>
<int_literal>   → <digit> { <digit> }
<bool_literal>  → true | false
<float_literal> → <int_literal> . <int_literal>
<char_literal>  → ' <letter> '
```

3. Token Table

For this project you MUST use the following token table (codes and descriptions must match).

Token#	Description	Token #	Description	Token #	Description
0	EOF	12	IF	24	SUBTRACT
1	INT_TYPE	13	ELSE	25	MULTIPLY
2	MAIN	14	WHILE	26	DIVIDE
3	OPEN_PAR	15	OR	27	BOOL_TYPE
4	CLOSE_PAR	16	AND	28	FLOAT_TYPE
5	OPEN_CURLY	17	EQUALITY	29	CHAR_TYPE
6	CLOSE_CURLY	18	INEQUALITY	30	IDENTIFIER
7	OPEN_BRACKET	19	LESS	31	INT_LITERAL
8	CLOSE_BRACKET	20	LESS_EQUAL	32	TRUE
9	COMMA	21	GREATER	33	FALSE
10	ASSIGNMENT	22	GREATER_EQUAL	34	FLOAT_LITERAL
11	SEMICOLON	23	ADD	35	CHAR_LITERAL

4. Error Table

For this project you MUST use the following error table (codes and descriptions must match). Optionally, you can add more specific errors by creating new codes between 18 and 99.

Error#	Description	Error#	Description
1	Source file missing	11	main expected
2	Couldn’t open source file	12	int type expected
3	Lexical error	13	']' expected
4	Digit expected	14	int literal expected
5	Symbol missing	15	'[' expected
6	EOF expected	16	identifier expected

7	'}' expected	17	';' expected
8	'{' expected	18	'=' expected
9	')' expected	19	identifier, if, or while expected
10	'(' expected	99	syntax error

5. Deliverables and Submission

Below is the list of minimum deliverables for this project:

- `parser.xxx` source code (e.g., `parser.py` or `parser.java`)
- `grammar.txt` file, and
- `slr_table.txt` file.

Files `grammar.txt` and `slr_table.txt` are only required if your parser is based on the shift-reduce (bottom-up) algorithm discussed in class. The format of those files must match the one used in class.

If you are writing your parser in a PL other than Python or Java you MUST provide specific instructions on how to properly setup your development environment, including IDE/compiler used (with version numbers) and how to compile/run your code. I should be able to test your code using MacOS. So if you are using a different platform I encourage you to contact me ahead of the deadline so I can properly set up my computer. If I cannot run your parser from the source code I cannot grade it!

Your source code MUST have a comment section in the beginning with the name(s) of the author(s) of the project. You are allowed to work together with another classmate. Teams of more than two students will NOT be accepted (NO exceptions). Only one of the members of the team needs to submit on Blackboard.

Please use ZIP format when submitting your project (no Z, RAR, or any other format will be accepted).

6. Running and Testing

For testing purposes, 15 source files are provided (download link here). Source files from 1-5 should be parsed without any errors. The table below summarizes the expected results when running the parser for each of the source files.

Source File	Expected Result
source1-5.c	Input is syntactically correct! Parse tree displayed.

source6.c	Error 12: int type expected
source7.c	Error 11: main expected
source8.c	Error 10: (expected
source9.c	Error 9:) expected
source10.c	Error 08: { expected
source11.c	Error 08: } expected
source12.c	Error 18: = expected
source13.c	Error 17: ; expected
source14.c	Error 10: (expected
source15.c	Error 19: identifier, if, or while expected

I should warn you that those tests are far from being comprehensive. The instructor reserves the right to run other tests on your code if deemed necessary. You are encouraged to create other tests on your own.

6. Rubric

This programming assignment is worth 100 points, distributed in the following way:

+3 command-line validation
+32 lexical analyzer works as expected
- +5 token codes match specification
- +1 recognizes EOF
- +3 recognizes identifiers
- +3 recognizes literals
- +5 recognizes special words
- +1 recognizes assignment operator
- +3 recognizes arithmetic operator
- +3 recognizes relational operators
- +2 recognizes logical operators
- +3 recognizes punctuators
- +2 recognizes delimiters
- +1 raises exception with proper error message when it fails

+60 syntax analyzer works as expected
- +10 grammar used matches specification
- +10 parser error codes match
- +20 parse tree is built correctly
- +30 parser properly shows syntactic errors

+5 submission follows instructions (student names identified in a comment section, zip format, source/grammar/slr table submitted, specific instructions provided when using different development platforms etc.)

10 points will be deducted for each day of late submission. I will not accept submissions that are five days (or more) late.