CST320

W2017

Lab2: Recursive Descent Calc

**Purpose:**

This lab will fulfill the following purposes:

1. Use a flex scanner as the input to a parser
2. Allow you to build a recursive descent parser for a simple language
3. Introduce you to error recovery processes for LL(1) parsers

**Process:**

Download the lab2.tar file from Blackboard. Unpack the file, but do NOT use the same working directory you used for Lab 1. We will get back to the Lab 1 directory next week.

Create a git repository similar to last week.

Create a recursive descent parser for the language defined below. Your parser will give a pass/fail indication on the program as well as syntax errors on failure. It will not generate code or evaluate expressions.

You will need to create/edit the following files:

1. Create a Makefile to build your project. Use last week’s as a starting point.
2. Create a flex generated scanner. Use last week’s as a starting point.
3. Write a Recursive Descent Parser. The files parse.cpp and parse.h can be used as starting points.

I want your recursive descent parser to follow the grammar given below exactly. Do not edit the grammar to make it “simpler”. You should have one function for each non-terminal with code for each production for that non-terminal.

**Testing your code**

Testing works as in last week’s lab. The first test contains a valid program. The second test contains a program with a variety of syntax errors. To pass the second test, your code must recover from syntax errors and continue parsing in an intelligent way. You can use semicolons as a synchronization point.

**Submitting your lab**

Label your final code with the label “lab2”. Provide the path to your git repository through Blackboard.

**Extra Credit**

The ‘correct’ file for this lab assumed very basic error reporting. For example, Line 4 reported the following error:

Found + when expecting a ‘;’ in line 4

It would be possible to provide better error messages. Looking at Line 4:

[1]+[2];

What went wrong? Did the programmer mean to use () instead of [] or did they mean to use a \* or / instead of a +?

I will provide up to 20% extra credit if you provide better error messages. If you want to earn this extra credit, AFTER labeling your regular submission in git, update your files, make a new commit, and label the new commit as “lab2 extra credit”. When you make your submission to Blackboard, indicate that you want me to grade your extra credit. I will pull your “lab2” submission and run the regression tests. I will then pull your extra credit version and compare your better error messages against the standard ones. If you want, you can also create a test3.lang to show-off your better error messages.

**Language definition**

The following grammar defines the language you are to parse for this lab:

PROG -> STMTS end

STMTS -> STMT STMTS

-> λ

STMT -> EXPR ;

EXPR -> (EXPR) EXPR’

-> TERM

EXPR’ -> PLUSOP (EXPR) EXPR’

-> λ

PLUSOP -> +

-> -

TERM -> [EXPR] TERM’

-> num

TERM’ -> TIMESOP [EXPR] TERM’

-> λ

TIMESOP -> \*

-> /

NOTES:

1. Your language should only support integers, not floating point numbers
2. You must allow C++ style comments
3. Your scanner must ONLY support the specified language. Points will be deducted if you include tokens from last week’s scanner that don’t belong in this lab.
4. I highly recommend you use the functions in utils.cpp for their intended purpose instead of trying to recode your own versions.