NOTE: Need to check tests against token definitions.

CST320

W2017

Lab 1: Scanner

**Purpose:**

This lab will fulfill the following purposes:

1. Introduce you to flex for building scanners
2. Reacquaint you with the Linux development environment
3. Introduce you to the git source code control tool
4. Work out the kinks in the lab submission process

**Process:**

Download the lab1.tar file from Blackboard. Copy the file to the Linux server and unpack the file using the following command:

tar –xvf lab1.tar

Remove the tar file from the directory.

Create a .gitignore file (note the leading period), which will inform git to ignore certain files. Add the following two lines to this file:

.gitignore

test/\*

Create a file with the name feedback.txt. The file must have the single line “Lab1 Feedback”. This can be done with the following command:

echo “Lab1 Feedback” > feedback.txt

The feedback.txt file will be used for me to give you feedback on your labs. I want it in your git repository so the feedback from week to week can be tracked.

Create and initialize a git repository using the following commands:

git init

git add \*

git status

git commit –m "Initial release"

NOTE: If this is your first time using git on our server, git may ask you to specify your user name or email. If so, follow the directions it gives you.

The git status command should show that the six files in your development directory are ready to be committed. The files in the test directory should not be included.

The lang.l file is the starting point for a flex input file. You need to edit this file so that flex can build a scanner for the language we will be building a compiler for. A list of all tokens is given below. Feel free to edit any of the other files as necessary.

The file tokens.h gives the values that flex must return for each token. You MUST use these values in order for your code to pass the regression tests.

**Testing your code**

The test directory contains regression tests. Each week you will get a new test suite for testing that week’s lab. To run the regression tests, run the following command from your development directory:

test/regress

You will get a list of tests and an indication of whether the test PASSED or FAILED. For this week, there is only one test. If a test fails, you can see what went wrong by manually running the test:

lang test/test1.lang out

diff out test/test1.correct

**Submitting your lab**

When you are done, you need to commit your changes to your git repository. It would also be a good practice to commit intermediate changes so that you can use git to rollback to a previous version in case you mess something up.

I will test your code by cloning (or pulling) your git repository. For this to be successful, you must fulfill two requirements:

1. You must have your completed lab committed to your git repository
2. You must attach a label to the version you want me to grade. The label will allow me to pull the version you want graded and will prevent subsequent commits from causing me to grade labn based on a partially completed labn+1. Use the command:

git tag lab1

to label the commit you want me to grade. When I grade your submission, I will examine the code labeled “lab1”, so be sure to use exactly this label.

NOTE: you must commit first and tag later. The git tag command attaches the label to the most recent commit.

When you are ready to submit your lab, provide the full path of your git repository through Blackboard. I ONLY need to know where your git repository is. You do NOT need to attach source code. I can pull everything I need from your repository.

If you use github (or some other publicly accessible git repository), I need the full path name to clone (or pull) your repository and I also need to full path name to a feedback.txt file on our server. I will give you feedback on our server, but will not place the feedback back into your repository. You are expected to place the feedback into your repository so that in the coming weeks I can see the feedback I gave you on earlier labs.

**Tokens**

The following tokens must be recognized by your scanner. Tokens from the first column should return the ASCII value of the token character. The tokens in the second, third, and fourth columns should return the appropriate value defined in tokens.h. Tokens from the fourth column have the normal meaning from C except that for float\_const, if there is a decimal point there MUST be a following digit. Integers and float constants do not include negative values.

In addition to the valid tokens, your scanner must remove whitespace and C++ (single line) style comments. If any invalid items are found, they must be reported as JUNK\_TOKEN (defined in tokens.h).

**Punctuation:**

(

)

{

}

[

]

;

,

.

+

-

\*

/

%

=

**Operators**

==

&&

||

**Keywords**

if

else

while

print

int

float

struct

return

array

**Others**

<identifiers>

<integer\_const>

<float\_const>