COVER PAGE
CS323 Programming Assignments

Fill out all entries 1 - 6. If not, there will be deductions!

Peer Review (Check one)

1. Names [1. Gregory Vasquez], (ThumbUP [**X**] or ThumbDown [])

   [2. Esteban Montelongo], (ThumbUP [] or ThumbDown [ ])

   [**if 3**.], (ThumbUP [] or ThumbDown [ ])

2. Assignment Number [Project 2]

3. Turn-In Dates: **Final Iteration with Documentation** [11/16/29]

4. Executable FileName [SyntaxAnalyzer.out]
   (**A file that can be executed without compilation by the instructor**)

5. LabRoom [CS 202]
(Execute your program in a lab in the CS building before submission)

6. Operating System/Language [Linux / c++]

To be filled out by the Instructor:

GRADE:

COMMENTS:

1. Problem Statement

The goal of this project was to create a syntax analyzer. The program reads in a file containing the source code where the lexical analyzer then parses it into tokens and lexemes. Where the Syntax analyzer then verifies if source code is syntactically correct. For this project we completed iterations one and two using a table driven parsers.

2. How to use your program

To compile the program it'll be the same for both iteration 1 and 2, just run the shell script (bash run.sh). Once that command is executed you will be prompted to type in the txt file. For the given syntax analyzer we have it running for the "in.txt" Type in that file name and hit enter. For simplicity we added a fold titled "Iteration12" that runs both iteration 1 and iteration 2 together in the same program but we also included the separate project folders if you wish to see them individually. For all of these, the commands to run them are all the same as above. After entering the file name and hit enter will first remove all files with extention .out and .o ; it will compile the .cpp files, create a linux executable named LexicalAnalyzer.out and it will execute the program.
Note: The shell script was created to create and compile changes faster. You can manually compile the files with (g++ main.cpp lexer.cpp syntax.cpp)

3. Design of your program

For iteration 1:
In this assignment we chose to take the table driven predictive parser approach. We first had to create a table using the First and Follow of our grammer. Once we fixed the left recursion, our productions were
Abbreviations:
St: Statement
As: Assignment
Ex: Expression
Ex' : Expression Prime
Te : Term
Te' : Term Prime
Fa : Factor

G = (N, T, S, R)
N = {St, As, Ex, Ex', Te, Te', Fa}
T = { id, num, +, -, *, /, (,), = , ;}
S = St
R = { St -> As
As -> id = Ex;
Ex -> TeEx'

Ex’ -> +TeEx’ | -TeEx’ | ε
Te -> FaTe’
Te’ -> *FaTe’ | /FaTe’ | ε
Fa -> (Ex) | id | num}

First(St) = {id} Follow(St) = {$}
First(As) = {id} Follow(As) = {$}
First(Ex) ={ (, id, num} Follow(Ex) = {; ,)}
First(Ex’) = {+, -, ε} Follow(Ex’) = {; ,)}
First(Te) = {(, id, num} Follow(Te) = { +, - , ; ,)}
First(Te’) = { *, / ,ε} Follow(Te’) = { +, - , ; ,)}
First(Fa) = { (, id, num} Follow(Fa) = { *, / ,+, - , ; ,)}

The table created from the information above is:

	id	num	+	-	*	/	(	)	=	;	$
St	As										
As	id=Ex;										
Ex	TeEx’	TeEx’					FaTe’				
Ex’			+TeEx’	-TeEx’				ε		ε	
Te	FaTe’	FaTe’					FaTe’				
Te’			ε	ε	*FaTe”	/FaTe”		ε		ε	
Fa	id	num					(Ex)				

For iteration 2:
Abbreviations:
St = Statement
DE = Declarative
TY = Type
MID = More Identifiers

G = (N, T, S, R)
N = {St, DE, TY, MID}
T = {id, int, float, bool, ; , comma}
S = St

R = { St -> DE
DE -> TY id MID; | ε
TY -> int | float | bool
MID -> comma id MID | ε }

First(St) = {int, float, bool, ε }
First(DE) = {int, float, bool, ε }
First(TY) = {int, float, bool}
First(MID) = {comma, ε }

Follow(St) = {$}
Follow(DE) = {$}
Follow(TY) = id
Follow(MID) = { ; }

	id	int	float	bool	;	,	$
ST		DE	DE	DE			ε
DE	ε	TY id MID;	TY id MID;	TY id MID;			ε
TY		int	float	bool			
MID					ε	, id MID	

4. Limitation

The output may be slightly different from the example output given in class but generally works for the given rules we constructed.
Text files(input sources) need to have spaces between each character to run with our program ie x = a;
This won't work x=a;

5. Shortcomings

None, that we know of.
Did not complete iteration three.