==================================

COMS W4115 Programming Languages and Translators

==================================

(Professor: Stephen A. Edwards)

Fantasy Football Stat Tracker Compiler

Michael Lam
Email: michael.lam@lmco.com
Phone: 215-815-5629
Due Date: 12/18/2007

Table of Contents

1 Introduction

My inspiration for designing a Fantasy Football Stat Tracker compiler transpired while participating in a fantasy football league with some friends. While the cost for participating in the fantasy football league is free of charge, most of the online fantasy football services charge a bundle for using their online stat tracker. The FFSTC will provide features for gathering player statistics, sorting based on category type, filtering by position type, stats comparison, and search by player name. Due to the development timeline, not all features will be available.

The features of the Fantasy Football Stat Tracker compiler will contain the following:

- Sorting algorithm for player stats
- Filtering for position by offense, defense, or specific position.
- Stats comparison by previous week or previous year
- Search function for players in the database
- Semantic checks for errors
    - Adding receiving stats to a defensive player will cause an error.
    - Adding field goal stats to a quarterback will be invalid.
    - Adding 1-point conversion stat to a quarterback will be invalid; however, a 2-point conversion will be valid.

2 Language Tutorial

FFSTC is a language used specifically for compiling statistical categories used in fantasy football leagues. When participating in a fantasy football draft, it is critical to obtain the key statistics for the top echelon players in the National Football League. This tutorial will show you how to write a simple program to obtain such critical statistics.

2.1 Commands

Prior to writing the language, here are some commands to get you started:

```
% java Main < [source file]
```

This will invoke the FFSTC compiler to begin compiling your source file.

2.2 FFSTC Database

The current version of FFSTC will use the statistics generated from the year 2006. Ensure that the following files are located in the Compiler directory:

QBStats.csv	Contains statistical categories for the top 50 quarterbacks for the year 2006. The statistical categories include (Name, Position, Team, *Ranking, Passing Yards, Passing Touchdowns, Passing Interceptions, Rushing Yards, and Rushing Touchdowns).
RBStats.csv	Contains statistical categories for the top 50 running backs for the year 2006. The statistical categories include (Name, Position, Team, *Ranking, Rushing Yards, Rushing Touchdowns, Receiving Yards, and Receiving Touchdowns).
WRStats.csv	Contains statistical categories for the top 75 wide receivers for the year 2006. The statistical categories include (Name, Position, Team, *Ranking, # of Receptions, Receiving Yards, Receiving Touchdowns, Return Yards, Return Touchdowns).
TEStats.csv	Contains statistical categories for the top 50 tight ends for the year 2006. The statistical categories include (Name, Position, Team, *Ranking, # of Receptions, Receiving Yards, and Receiving Touchdowns).
KKRStats.csv	Contains statistical categories for the top 34 Kickers for the year 2006. The statistical categories include (Name, Position, Team, *Ranking, Field Goals Made for 0-19, 20-29, 30-39, 40-49, and 50+ yards, 1PT Made).
DEFStats.csv	Contains statistical categories for all 31 teams for the year 2006. The statistical categories include (Name, Position, Team, Rank, Points Allowed, Sacks, Safety, Interceptions, Fumble Recovery, Touchdowns).

** Rankings are based on Yahoo Rankings*

Here are some fields extracted from the input file. All categories must begin with the token “%”:

```
%Name,%Position,%Team,%Rank,%PassYD,%PassTD,%PassINT,%RushYD,%RushTD
%Peyton Manning,%QB,%IND,%4,%4397,%31,%9,%36,%4
%Drew Brees,%QB,%NO,%9,%4418,%26,%11,%32,%0
%Michael Vick,%QB,%ATL,%10,%2474,%20,%13,%1039,%2

%Name,%Position,%Team,%Rank,%RushYD,%RushTd,%RecYD,%RecTD
%LaDainian Tomlinson,%RB,%SD,%1,%1815,%28,%508,%3
%Larry Johnson,%RB,%KC,%2,%1789,%17,%410,%2
%Steven Jackson,%RB,%STL,%3,%1528,%13,%806,%3
```

2.3 FFSTC Example

The following is an example of the syntax in the main program. The language should always begin with the keyword “ffstcmain” and end with the keyword “ffstcend.”

```
ffstcmain myprogram

…
…

ffstcend
```

Within the main program, you can write user-define functions using the following syntax. The keyword “statsfunc” is used to identify a function, following by open and close parenthesis. “statsint” is a keyword to define an integer “q” as an argument in the function. The keywords “subbegin” and “subend” must be included to enclose the function body.

```
ffstcmain myprogram

statsfunc myfunction (statsint q)
      subbegin myfunctionstatements
            …
      subend

ffstcend
```

In addition to defining functions, you can write statements and expressions within the main program. Prior to writing the statements and expressions, the keywords “subbegin” and “subend” must be enclosed.

```
ffstcmain myprogram

statsfunc myfunction (statsint q)
      subbegin myfunctionstatements
            …
      subend

subbegin mystatements
      z = {x + y * w};
      if {z > 10}
         x = {z};
      endif;
subend

ffstcend
```

2.4 FFSTC Comments

Comments in FFSTC are omitted during the compilation process. You may designate comments by using the single-line notation “@@”

```
@@ This is a comment.
```

Or you may designate comments by using the multi-line notation “@%” and “%@”

```
@%
This is
also
a comment.
%@
```

3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 Tokens

There are six types of tokens in this language: identifiers, keywords, constants, strings, expression operators, and separators. A whitespace must be used to separate tokens.

3.1.2 Comments

FFSTC supports single and multi-line comments. Single line comments are denoted using the notations @@. Multi line comments begin with @% and ends with %@

Example:

```
@@ This is a valid comment

@%This is also a
valid comment%@
```

3.1.3 Identifiers

An identifier is a sequence of letters and digits with the first character beginning with an alphabet. Identifiers are case sensitive.

3.1.4 Keywords

The following are identifiers reserved as keywords and may not be used otherwise.

while	endwhile	QB	TeamName
if	else	RB	bool
endif	bool	WR	int
FALSE	TRUE	TE	
for	endfor	KKR	
continue	break	DEF	

The following are identifiers reserved for designating team names.

BUF	BAL	HOU	DEN
MIA	CIN	IND	KAN
NE	CLE	JAC	OAK
NYJ	PIT	TEN	SD
DAL	CHI	ATL	ARI

NYG	DET	CAR	SF
PHI	GB	NO	SEA
WAS	MIN	TB	STL

3.1.5 Constants

Three types of constants are allowed in FFSTC: integer, double, and string. All constants are capitalized.

Example:

const MAXYARDS = 100;

3.1.6 Spatial Delimiters

Spatial delimiters in FFSTC are defined in the following table.

(	)	{	}
[	]	;	:
,	.		

3.1.7 Operator Precedence

The following operators can be used in FFSTC and the following table indicates the operator precedence from highest to lowest order. The operators with equal precedence will be evaluated using the left to right associativity.

() {} []	Parenthesis, Braces, and Bracket Operators
/ *	Multiplicative and Division Operators
+ -	Additive and Subtraction Operators
== > >= < <= !=	Relational Operators
!	Negation Operator
&& ||	Logical Operators
= /= *= += -=	Assignment Operators

3.1.8 Data Types

The following table shows the data types in FFSTC:

QB	int
RB	bool
KKR	
TE	
DEF	
TeamName	

3.2 Declarations

Declaring variables in FFSTC can take the following form:

```
QB myQuarterBack;
RB myRunningBack;

QB myQBRankings[10];
RB myRBRankings[10];
```

3.2.1 QB Type

The QB type will be associated with the statistical categories passYards, passTD, and passINT.

3.2.2 RB Type

The RB type will be associated with the statistical categories rushYards, rushTD.

3.2.3 WR Type

The WR type will be associated with the statistical categories rcvYards, rcvTD.

3.2.4 TE Type

The RE type will be associated with the statistical categories rcvYards, rcvTD.

3.2.5 KKR Type

The KKR type will be associated with the statistical categories FGMade, FGMiss.

3.2.6 DEF Type

The DEF type will be associated with the statistical categories defSACK, defINT.

3.3 Expressions

3.3.1 Mathematical Expressions

In FFSTC, expressions can be written to add statistical categories.

3.3.1.1 Multiplicative Expressions

```
multiplicative-expression
        : attribute-expression
        | multiplicative-expression * postfix-expression
```

3.3.1.2 Divisions Expressions

```
Division-expression
        : attribute-expression
        | Division-expression * postfix-expression
```

3.3.1.3 Additive Expressions

```
additive-expression: multiplicative-expression
        | additive-expression + multiplicative-expression
```

3.3.1.4 Subtraction Expressions

```
Subtraction-expression: multiplicative-expression
        | additive-expression - multiplicative-expression
```

3.3.2 Relational Expressions

```
relational-expression
        : pack-expression
        | relational-expression < pack-expression
        | relational-expression > pack-expression
        | relational-expression <= pack-expression
        | relational-expression >= pack-expression
        | true
        | false
```

3.3.3 Logical Expressions

```
conditional-expression: logical-OR-expression
        logical-OR-expression: logical-AND-expression
        | logical-OR-expression || logical-AND-expression
        logical-AND-expression: equality-expression
        | logical-AND-expression && equality-expression
        equality-expression: relational-expression
        | equality-expression == relational-expression
        | equality-expression != relational-express
```

3.4 Statements

All statements in FFSTC must be written in the program body enclosed by "subbegin" and "subend."

3.4.1 Conditional Statements

The following syntax denotes the *if* statement:

```
if (expression)
{
        statement
}
else if (expression)
{

}
endif
```

3.4.2 Looping Statements

The following syntax denotes the while statement:

```
while (expression)
{
        statement
}
endwhile
```

3.4.3 Print Statements

The printing for scoring statistics are handled through the build-in function *FFLPrint(x,y).* Users may only use print statements to echo messages.

For Example:

```
printf("Hello World.");
```

3.5 Scope

In FFSTC, all variables declared within the body of a function or statements are static. Variables declared prior to the program body will be global.

3.6 Built-in Functions

All Built-in functions in FFSTC begin with FFL.

FFLGetDatabase();	Required in the beginning of program. Without this declaration, FFSTC will not be able to obtain records.
bool FFLSearch(*Para1*,*Para2*,*Para3*,*Para4*);	Search for the statistics for a specified position type by name and team and stores the value for position type. Returns true or false. *Para1* ➔ *Position Type* *Para2* ➔ *Lastname* *Para3* ➔ *Firstname* *Para4* ➔ *Team Identifier* For Example: QB myQB; bool isFound (false); isFound = FFLSearch(myQB, “Favre”, “Bret”, GB);
int FFLRetrieve(*Para1,Para2,Para3,Para4* *);*	Retrieve the statistics for a specified position type ranked highest by the statistic category type. The record will be stored into an array and the record with the highest ranking will be returned. *Para1* ➔ *Position Type* *Para2* ➔ *Position Identifier* *Para2* ➔ *statistic category* *Para3* ➔ *number of records* *Para4* ➔ *Team Identifier* Example: QB myQBRankings[11]; FFLRetrieve(myQBRankings, QB, passYards, 10);
FFLPrint(*Para1, Para2*);	Prints the statistics for a specified position type by the entire 2006 season or weekly. *Para1* ➔ *Position Type* *Para2* ➔ *0 denotes entire season, 1-17 denotes weeks 1 through 17.* Example: @@ Prints my quarterback statistics for the entire 2006 season FFLPrint(MyQB, 0); @@ Prints my quarterback statistics for week 6 FFLPrint(MyQB, 6); Options: Prints the statistics for a specified TeamName type by weekly only. Example: @@ Prints team scoring totals for week 11 FFLPrint(myCustomTeam, 11);
TeamName Type = FFLFormulateTeam (*Para1*, *Para2*, *Para3*, *Para4*, *Para5*, *Para6*, *Para7*);	In a fantasy football league, a team consists of seven position types: QB, RB, WR, WR, TE, KKR, and DEF. This build-in function will create a team using specified position types and assign it to TeamName Type. *Para1* ➔ *QB Type* *Para2* ➔ *RB Type* *Para3* ➔ *WR Type 1* *Para4* ➔ *WR Type 2* *Para5* ➔ *TE Type* *Para6* ➔ *KKR Type* *Para7* ➔ *DEF Type* Example: TeamName myCustomTeam; myCustomTeam = FFLFormulateTeam(myQB, myRB, myWR1,myWR2, myTE, myKKR, myDEF);

3.7 User-Defined Functions

Functions in FFSTC can be defined by users using the following C-like format:

```
<return type>? <identifier> (<Para1>, <Para2>, <Para3>, ...)
{
        body
};
```

The return type can either be the following data types:

QB	TE	Int
RB	DEF	Bool
KKR	TeamName	

Para1, Para2, and Para3 represent function parameters. The function must end with a semi-colon.

3.8 Sample Program

```
@@ #FFSTC indicates the beginning of the program
#FFSTC
@@ Required declaration to obtain scoring statistics for all position types
FFLGetDatabase();

QB myQB;
RB myRBRankings[11];
WR myWRRankings[11];
RB myRB;
WR myWR1;
WR myWR2;
TE myTE;
KKR myKicker;
DEF myDefense;
bool isFound(false);
int RBHigh, WRHigh;

@@ Search for the top ten rankings amongst position type based on statistical category
RBHigh = FFLRetrieve(myRBRankings, RB, rushYards, 10);
WRHigh = FFLRetrieve(myWRRankings, WR, rcvYards, 10);

@@ Print the top ten rankings
for (int i=0;i<10;i++)
        FFLPrint(myQBRankings[i], 0);
endfor

@@ Assign my players to the highest ranked position player
myRB = myRBRankings[RBHigh];
myWR = myWRRankings[WRHigh];

@@ Print the seasonal stats for my players.
FFLPrint(myRB, 0);
FFLPrint(myWR, 0);

@@ Search for my starting quarterback
isFound = FFLSearch(myQB, “McNabb”, “Donovan”, PHI);

if (isFound)
        FFLPrint(myQB, 0);
endif

@@ Denotes the end of the program
#ENDFFSTC
```

4 Project Plan

4.1 Project Timeline

September week 4	Proposal
October week 1	ANTLR setup
October week 3	LRM
November week 2	Lexer development
November week 3	Parser development
November week 4	Interface Components
December week 1	Testing
December week 2	Regression Testing
December week 3	Project Documentation

4.2 Roles and Responsibilities

Michael Lam: Lexer, Parser and Treewalker, and documentation

4.3 Software Development Environment

The project was developed using the CYGWIN environment version 3.7-1 and Java version 1.4.2_16. For configuration management, CVS version control system was used.

4.4 Project Log

September 25, 2007:	Submitted Proposal
October 18, 2007:	Submitted LRM
November 2, 2007:	Installed CYGWIN and setup environ paths
November 2, 2007:	Compiled ANTLR version 2.7.7 and setup environ paths
November 29, 2007:	Developed Parser and Lexer for stats input database
December 15, 2007:	Updated Project Documentation

5 Architectural Design

The ffstc interpreter has the following components:

Lexer:	scanning user inputs and program files into tokens
Parser:	syntax analysis of program and AST conversion
Tree Walker:	AST processing.

6 Test Plan

A Main program and an input file is created to test the parser and lexer.

The following command is used to compile ffstc:

```
% java antlr.Tool ffstc.g
```

After the java files have been generated, the following command compiles the java files:

```
% javac *.java
```

The following command starts the test:

```
% java Main < input.dat
```

After debugging errors, the following is printed:

```
ffstcmain myprogram statsint { w = 3 } ; statsint { x = 22 } ; statsint { y = 20 } ; statsint { z = 0 }
; statsfunc myfunction ( statsint q ) statsint { p = 24 } ; subbegin myfunctionstatements q = 3 ;
subend subbegin mystatements z = ( + x ( * y w ) ) ; if ( > z 10 ) x = z ; endif ; subend ffstcend
```

Main.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class Main {
   public static void main(String[] args) {
        try {
            DataInputStream input = new DataInputStream (System.in);
            ffstcLexer lexer = new ffstcLexer(input);
            ffstcParser parser = new ffstcParser(lexer);
            parser.program();
            CommonAST parseTree = (CommonAST)parser.getAST();
            System.out.println(parseTree.toStringList());
            ffstcTreeWalker walker = new ffstcTreeWalker();
            double r = walker.expression(parseTree);
            System.out.println("Value: "+r);

        } catch(Exception e) {
            System.err.println("exception: "+e);
        }
    }
}
```

input.dat

```
@@This is a comment
```

```
@%
This is also a comment
%@
@@ The following expressions are omitted
@@(3+4)*5/7*33+4;
@@(22+4-43*4/(99-43)*4);
@@(5*(2+3));

ffstcmain myprogram

statsint {w = 3};
statsint {x = 22};
statsint {y = 20};
statsint {z = 0};

      statsfunc myfunction (statsint q)
      statsint {p=24};
            subbegin myfunctionstatements
                  q = {3};
            subend

subbegin mystatements
      z = {x + y * w};
      if {z > 10}
         x = {z};
      endif;
subend

ffstcend
```

Example html output file:

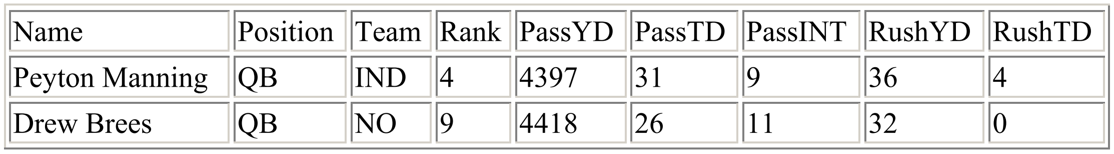
```
<html>
<head>
<title>FFSTC Version 1.0.0 File Output</title>
</head>
<body>
<table border="1">
<tr>
      <td>Name</td>
      <td>Position</td>
      <td>Team</td>
      <td>Rank</td>
      <td>PassYD</td>
      <td>PassTD</td>
      <td>PassINT</td>
      <td>RushYD</td>
      <td>RushTD</td>
   </tr>

<tr>
      <td>Peyton Manning</td>
      <td>QB</td>
      <td>IND</td>
      <td>4</td>
      <td>4397</td>
      <td>31</td>
      <td>9</td>
      <td>36</td>
      <td>4</td>
   </tr>

<tr>
      <td>Drew Brees</td>
      <td>QB</td>
      <td>NO</td>
      <td>9</td>
      <td>4418</td>
      <td>26</td>
      <td>11</td>
      <td>32</td>
      <td>0</td>
   </tr>

<tr>
```

Example html output file (Browser View):



Name	Position	Team	Rank	PassYD	PassTD	PassINT	RushYD	RushTD
Peyton Manning	QB	IND	4	4397	31	9	36	4
Drew Brees	QB	NO	9	4418	26	11	32	0

7 Lessons Learned

An important lesson that I learned while working on this project is that writing a compiler can be very challenging when the specifications are not clearly defined, especially for someone working on this project individually. This is definitely a learning experience considering that java is not my primary language of expertise. As a CVN student, it was detrimental that I was not able to interact with other students. I would recommend a student taking this class in the future to be on campus.

7.1 Advice for future teams

I would recommend students taking this class in the future to be on campus because the interaction with students, teaching assistants, and professors can be critical when working on a project of this magnitude.

8 Appendix

8.1 Code Listing for each module

Main.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class Main {
   public static void main(String[] args) {
        try {

            FileInputStream fileInput = null;
            DataOutputStream fileOutput = null;

            for (int i=0;i<=5;i++)
            {
                  String statsInputFile = "";
                  String statsOutputFile = "";

                  if (i==0)
                  {
                        statsInputFile = "QBStats.csv";
                        statsOutputFile = "QBStats.html";
                  }
                  else if (i==1)
                  {
                        statsInputFile = "RBStats.csv";
                        statsOutputFile = "RBStats.html";
                  }
                  else if (i==2)
                  {
                        statsInputFile = "WRStats.csv";
                        statsOutputFile = "WRStats.html";
                  }
                  else if (i==3)
                  {
                        statsInputFile = "TEStats.csv";
                        statsOutputFile = "TEStats.html";
                  }
                  else if (i==4)
                  {
                        statsInputFile = "KKRStats.csv";
                        statsOutputFile = "KKRStats.html";
                  }
                  else if (i==5)
                  {
                        statsInputFile = "DEFStats.csv";
                        statsOutputFile = "DEFStats.html";
                  }
                  try {
                        fileInput = new
FileInputStream(statsInputFile);
                        fileOutput = new DataOutputStream(new
FileOutputStream(statsOutputFile));
                  } catch(Exception e) {
                        System.err.println("exception: "+e);
                  }

                  DataInputStream statsinput = new DataInputStream
(fileInput);
                  ffstcCSVLexer CSVlexer = new
ffstcCSVLexer(statsinput);
                  ffstcCSVParser CSVparser = new
ffstcCSVParser(CSVlexer);

                  String ffstcOutput = "";
                  ffstcOutput = "<html>\n";
                  ffstcOutput += "<head>\n";
                  ffstcOutput += "<title>FFSTC Version 1.0.0 File
Output</title>\n";
                  ffstcOutput += "</head>\n";
                  ffstcOutput += "<body>\n";
                  ffstcOutput += CSVparser.file();
                  ffstcOutput += "</body>\n";
                  ffstcOutput += "<html>\n";
                  fileOutput.writeBytes(ffstcOutput);
                  fileOutput.close();
            }

            DataInputStream input = new DataInputStream (System.in);
            ffstcLexer lexer = new ffstcLexer(input);
            ffstcParser parser = new ffstcParser(lexer);
            parser.program();

            CommonAST parseTree = (CommonAST)parser.getAST();
            System.out.println(parseTree.toStringList());
//          ASTFrame frame = new ASTFrame("The tree", parseTree);
//          frame.setVisible(true);

            ffstcTreeWalker walker = new ffstcTreeWalker();
            double r = walker.expression(parseTree);
            System.out.println("Value: "+r);

        } catch(Exception e) {
            System.err.println("exception: "+e);
        }
    }
}
```

8.2 Source files (.g files)

ffstc.g

```
/*
* ffstc.g : ANTLR grammar.
*
* ffstcLexer and ffstcParser
*
* @author Michael Lam - ml2882@columbia.edu
*
*/

// FFSTC Parser
class ffstcParser extends Parser;
options
{
      k = 2;
      buildAST = true;
}

program
      : "ffstcmain" ID
            subprogram
      "ffstcend"
      ;

subprogram
      : (ffstcvarDecl)*
        (ffstcfuncDecl)*
      "subbegin" ID
            (statement)*
      "subend"
      ;

ffstcvarDecl
      : "statsint" LBRACE ID ASSIGN INT RBRACE
      (ASSIGN INT)?
      SEMI
      ;

ffstcfuncDecl
      : "statsfunc" ID (funcParam)?
            subprogram
      ;

funcParam
      : LPAREN paramSpec (COMMA paramSpec)* RPAREN
      ;

paramSpec
      : ("statsint")? ID
      ;
```

```
statement
      : ifStatement
      | assignStatement
      | (endStatement) => endStatement
      ;

ifStatement
      : "if" ifbody "endif" SEMI
      ;

ifbody
      : expression
      (statement)*
      ("elseif" ifbody
      | "else" (statement)*
      )?
      ;

assignStatement
      :
      varRef ASSIGN expression SEMI
      ;

endStatement
      :
      SEMI
      ;

varRef
      : ID
      (expression)*
      ;

expression  : LBRACE! sumExpr RBRACE! ;
sumExpr     : prodExpr ((PLUS^|MINUS^) prodExpr)* ;
prodExpr    : powExpr ((MULT^|DIV^|MOD^) powExpr)* ;
powExpr     : atom (POW^ atom)? ;
atom        : INT | ID | expression;

// FFSTC Lexer
class ffstcLexer extends Lexer;
options {
  k=2; // Set the Lookup value
  // Define the set of Unicode characters that characters in the
inputstream must belong to
  charVocabulary = '\3'..'\377';
  testLiterals=false;    // Don't automatically test for literals
}

WS  :   (   ' '
        |   '\t'
        )+
        {$setType(Token.SKIP);} //ignore this token
    ;

NWLN :  (('\r''\n') => '\r''\n'  //DOS
        | '\r'                  //MAC
        | '\n')                  //UNIX
        { newline();$setType(Token.SKIP);}
        ;

RECORD  : '%'! (~(','|'\r'|'\n'))+ ;

// FFSTC single line and multi line comments
COMMENT : ( "@%" (
      options {greedy=false;}
      :
      (NWLN)
      | ~( '\n' | '\r' )
      )* "%@"
      | "@@" (~( '\n' | '\r' ))* (NWLN)
      ) { $setType(Token.SKIP); }
;

LPAREN : '(';
RPAREN : ')';
LBRACE : '{';
RBRACE : '}';
LBRACK : '[';
RBRACK : ']';

MULT : '*';
PLUS : '+';
MINUS : '-';
DIV : '/';
MOD : '%';
POW: '^';

// FFSTC Literals
protected DIGIT : '0'..'9';
INT : (DIGIT)+ ;

protected LETTER : 'a'..'z' | 'A'..'Z';

// FFSTC Identifiers
ID options {testLiterals = true;}
      : LETTER (LETTER|DIGIT)*
      ;

// FFSTC Operators
COMMA : ',';
SEMI : ';';
COLON : ':';
ASSIGN : '=';
PLUSEQ : "+=";
MINUSEQ : "-=";
MULTEQ : "*=";
RDVEQ : "/=";
GE : ">=";
LE : "<=";
GT : '>';
LT : '<';
EQ : "==";
NEQ : "!=";
```

```
TRSP : '\'';

{
      import java.io.*;
      import java.lang.Math;
}

class ffstcTreeWalker extends TreeParser;

expression returns [double rval]
  { double x,y; rval=0; }

  : #(PLUS  x=expression y=expression)  { rval=x+y; }
  | #(MINUS x=expression y=expression)  { rval=x-y; }
  | #(MULT  x=expression y=expression)  { rval=x*y; }
  | #(DIV   x=expression y=expression)  { rval=x/y; }
  | #(MOD   x=expression y=expression)  { rval=x%y; }
  | #(POW   x=expression y=expression)  { rval=Math.pow(x,y); }
  | #(LBRACE x=expression) { rval=x;}
  | i:INT { rval=(double)Integer.parseInt(i.getText()); }
;
```

ffstcCSV.g

```
/*
* ffstcCSV.g : ANTLR grammar.
*
* ffstcCSVLexer and ffstcCSVParser
*
* @author Michael Lam - ml2882@columbia.edu
*
*/

// FFSTC Parser
class ffstcCSVParser extends Parser;
options
{
      k = 2;
      buildAST = true;
}

file returns[String table = new String()]
      {String lineData;int i=1; table+="<table border=\"1\">\n";}
      : ( lineData=line {table+=lineData;table+="\n";}
            (NWLN lineData=line {table+=lineData;table+="\n";i++;} )*
            (NWLN)? EOF )
            {table+="</table>";}
            {System.out.println(i+ " lines matched\n");}
      ;

line returns [String lineData = new String()]
      {String recordData; lineData+="<tr>\n";}
      : ( (recordData=record {lineData+=recordData;}
        )+ )
        {lineData+="   </tr>\n";System.out.println("Line matched\n");}
      ;

record returns [String recordData = new String()]
      {recordData+=("      <td>");}
      : ( (r:RECORD) (COMMA)? )
      {recordData += (r.getText());
       recordData += "</td>\n";}
      ;

// FFSTC Lexer
class ffstcCSVLexer extends Lexer;
options {
  k=2; // Set the Lookup value
  // Define the set of Unicode characters that characters in the
inputstream must belong to
  charVocabulary = '\3'..'\377';
  testLiterals=false;    // Don't automatically test for literals
}

WS  :   (   ' '
        |   '\t'
        )+
        {$setType(Token.SKIP);} //ignore this token
    ;

NWLN :  (('\r''\n') => '\r''\n'  //DOS
        | '\r'                  //MAC
        | '\n')                  //UNIX
        { newline();}
        ;

RECORD  : '%'! (~(','|'\r'|'\n'))+ ;

// FFSTC Literals
protected DIGIT : '0'..'9';
INT : (DIGIT)+ ;

protected LETTER : 'a'..'z' | 'A'..'Z';

// FFSTC Identifiers
ID options {testLiterals = true;}
      : LETTER (LETTER|DIGIT)*
      ;

// FFSTC Operators
COMMA : ',';
```