TableGen
Final Report

Andrey Falko (asf2125) Daniel Pestov (dp2297) Del Slane (djs2160)
Timothy Washington (tw2212)

December 18, 2007

Contents

Chapter 1

Introduction

1.1 Introduction

The rapid creation of algorithmically complicated data tables is currently not facilitated to a useful extent by current software such as Excel. Spreadsheet applications allow for the writing of algorithms, but it is both time consuming and cumbersome. Inputting data by tabbing through cells alone takes a long time. Data in a spreadsheet is not presentable because it is used to generate the final numbers.

TableGen will be the solution to these problems: it will be a powerful interpreted programming language with enough power to specify complex functions which can transform data. It will have an efficient syntax for assigning values to cells in a data table without any clicking, scrolling, or tabbing. Users will be able to utilize basic data structures such as arrays to easily list their data and to subsequently and conveniently compute the final results. All of this functionality will be implemented in a clean manner – unnecessary punctuation will be excluded from the language to make it less of a burden to program with.

The purpose of TableGen is to work on large input and to generate large output. It is not efficient to use TableGen to conduct a computation over x variables and output one value at the end. All output in TableGen is made into cells of a table, so if you are not planning to make a table, don't use TableGen.

1.2 General Plan

We plan to implement a language whose syntax and semantics help people fill the cells of a table or tables with their data. Our language will allow people to:

- Fill cells, rows, or columns with minimal typing.

- Fill cells, rows, or columns by manipulating existing data.
- Output plain text, TeX/LaTeX, HTML tables.
- Allow the user to make "output modules" so that they can specify output organization other than HTML, LaTeX, etc (in language the compiler will be written in).
- Allow user to utilize statistical functions such as average, standard deviation, etc.
- Allow the user to easily add statistical functions and other functions (in our language).

1.3 Technical Plan

We plan to write the interpreter for the language mostly in java (backend and most of the frontend). The other parts will be in Perl (frontend). Java makes sense for a couple of reasons:

1. All of us know how to code in it.
2. Antlr is a java application, so it would be natural to utilize the same language. Perl makes sense for part of the frontend because it much easier to control the java binaries with it and to implement various command-line options with it.

The testing infrastructure will be written using a combination of Make and Perl. We plan to use Perl for the following reasons:

1. The testing infrastructure will require a lot of string comparisons, something Perl is very good at.
2. Most of us know how to code in it.

1.3.1 Language Design

Our language will have white space syntax. Our goal is to make this language require minimal typing on the user's end and require minimal memorization. To accomplish this we have the following features:

- Data Types

  There are two data types in TableGen: strings and numbers. All variables in TableGen are hash-arrays, including those that have a single element. Users can specify types by doing:

```
num:var1 = 1, 2
str:var2 = "one", "two"
```

- Data Entry

  TableGen allows users to enter data into a set of coordinates by using the put-in operator (`->`):

  ```
  var1 = "Hello"

  var1 -> 0,0
  ```

  Prints:

  ```
  Hello
  ```

- Subroutines and Control Structures

  TableGen allows subroutines:

  ```
  func compute num:a num:n
          return a * n

  compute (3, 4) -> 0, 0
  ```

  Prints 12.

  TableGen allows the standard while, if, for, and foreach control structures.

- Special Variables and Functions

  TableGen has special functions that allow users to do various things. For example, there is `_tail`, which returns the last element of a list. There are others:

  Some of the reserved functions:

  - `_avg (list)`: Average
  - `_tail (list)`: Last element of list
  - `_length (list)`: Length of list
  - `_sum (list)`: Sum elements of list

Chapter 2

Language Tutorial

2.1 Preparing TableGen

The first step is to download the source code. We assume that you downloaded `TableGen.tar.bz2`. After that extract the code:

```
tar -xjf TableGen.tar.bz2
```

Make sure that you have make, Java, and Perl installed on your system. Also make sure that Antlr3 is in your `CLASSPATH` environmental variable. If you have these prerequisites proceed to compile:

```
cd TableGen
make compile
```

2.2 A Simple Program

Lets take the following simple program and save it in a file called `hi.tg`:

```
variable = "Hi, there"
variable -> 0,0
```

To run the program, execute the main TableGen executable:

```
./TableGen hi.tg
```

If you'd like to get html output do this:

```
./TableGen --output-type html hi.tg
```

Likewise you can get a latex table:

```
./TableGen --output-type latex hi.tg
```

If you'd like to customize your output even more, execute TableGen with the `--help` option:

```
./TableGen --help
```

2.3 A More Complicated Program

The more complicated program collects statistics about basketball players on the same team. It compares the basketball players and determines how many points per game each score when the other players play. Let's take things a few lines at a time. First write some comments and define some data:

```
/* This program returns statistics regarding how well players on the
* New Jersey Nets play with each other.
*/

// Negative number means that the player missed the game.
Kidd = 7, 2, 16, 9, 6, 17, 14, 19, 11, 16, 2, 12, 14, 15, 12, 19, 6, 7, -1, 10
Carter= 24, 7, 18, 12, 24, 17, -1, -1, -1, -1, -1, 13, 15, 19, 32, 24, 22, 19, 19, 32
Jefferson = 29, 27, 22, 25, 25, 28, 32, 21, 15, 18, 22, 30, 30, 27, 27, 23, 21, 36, 31, 29
```

The first few lines constitute a multi-line comment. Be line before data definition is a single-line comment. The next three lines define three lists. All elements are added to the list through comma separators. Note that TableGen automatically figures out what is a in integer and what is a double.

```
str:players = "Jason Kidd", "Vince Carter", "Richard Jefferson"
```

This line defines a string. TableGen allows users to explicitly specify weather something is a number or a string.

```
num:row = 0
num:col = 1
foreach player in players
        player -> row,col
        player -> col,row
        col = col + 1
```

These lines print text into our first column and first row. The output of these lines will be this:

```
                  Jason Kidd        Vince Carter      Richard Jefferson
Jason Kidd
Vince Carter
Richard Jefferson
```

Note TableGen's use of the put-in operator (->). The put-in operator tells TableGen's interpreter to put data into a set of coordinates. When programming in TableGen always remember that the first coordinate is up-and-down, while the second coordinate is side-to-side. Also, don't try negative coordinate because TableGen will yell at you. Also notice how TableGen know about blocks. TableGen needs indentation for specification of blocks. This indentation can only be made using tabs, so forget about the space bar when indenting in TableGen.

```
printPPG (Kidd 1)
printPPG (Carter 2)
printPPG (Jefferson 3)

compare (player1:Kidd coord1:1 player2:Carter coord2:2)
compare (coord2:3 player1:Kidd player2:Jefferson coord1:1)
compare (Carter 2 Kidd 1)
compare (Carter 2 Jefferson 3)
compare (Jefferson 3 Kidd 1)
compare (Jefferson 3 Carter 2)
```

These are lines are function calls to subroutines. Note that unlike other languages, TableGen does not use commas to separate function parameters; is uses spaces instead. Notice another thing about TableGen, parameter names can be passed in function calls. By passing parameters with their names attached, you can call function with parameters in any order.

```
func printPPG player coord
        num:totalPoints = 0
        num:gamesPlayed = 0
        // Find per game average
```

```
        foreach score in player
                if (score >= 0)
                        totalPoints = totalPoints + score
                        gamesPlayed = gamesPlayed + 1

        _round (totalPoints / gamesPlayed 10.0) -> coord, coord

func compare player1 coord1 player2 coord2
        num:totalPoints = 0
        num:gamesPlayed = 0

        // If both played
        for i = 0; i < _length (player1); i = i + 1
                if player1[i] >= 0 && player2[i] >= 0
                        totalPoints = totalPoints + player1[i]
                        gamesPlayed = gamesPlayed + 1

        _round (totalPoints / gamesPlayed) -> coord1,coord2
```

This section of code contains two function or subroutine definitions. The first one totals of the score of all the sames a player played and returns the scoring average. The second function averages the total score of a player when another player played. Notice our use if reserved functions such as `_round`. The `_round` subroutine can be used in two ways, the user can choose the default rounding at the hundredth position or supply an extra parameter and tell TableGen what the rounding factor should be. In our case, we chose 10.0 for demonstration purposes.

2.4 A Mathematical Program

The following program recursively creates the Pascal triangle. Like the last one, let's take it section-by-section.

```
/* Factorial algorithm using a cache to compute factorials. */
num:factCache = 1
func factorial num:n num:factCache
        if n < _length (factCache)
                return factCache[n]
        prod = _tail (factCache)
        while (_length (factCache) <= n)
                /* Append prod onto end of factCache array. */
                prod = prod * _length (factCache)
                factCache = factCache, prod
        return prod
```

This is a factorial function that uses a cache of computed value for optimization. First notice that we pass the global variable, `factCache` as a parameter to the factorial function this necessary because functions are designed not to see global variables. Another thing to note is the line `factCache = factCache, prod`. This line is the way to concatenate lists together. In this case `factCache` is having `prod` added to the end of it.

```
func pascalCombination num:n num:k
        return factorial (n factCache) / (factorial (k factCache) * factorial ((n-k) factCache))
```

These to lines define the pascal combination function. Notice how TableGen allows you to save space by being able to pass computation to function calls (`factorial ((n-k) factCache)`).

```
max = 10 /* Set maximum # of rows to output */

i = 0
j = 0
while (i < max)
        while (j <= i)
                pascalCombination (i j) -> i, j
                j = j + 1
        i = i + 1
        j = 0
```

Here, we output a triangle containing values of the pascalCombination. This part of the code is fairly straight-forward. The output of this example is:

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
```

2.5 Fun With Hash Keys

One of the main features of TableGen is that variables can act as hashes and arrays at the same time. Take this short program for example:

```
people = "American">"Betty", "Russian">"Vladimir", "German">Konstantine

func printTheGerman
people.German -> 0,0

printTheGerman (people)
```

Chapter 3

Language Manual

3.1 Introduction

TableGen is an interpreted programming language designed to allow programmers to quickly generate large tables. TableGen allows experienced users to enter large data more efficiently than when using graphical spreadsheet applications, especially when data is generated via algorithmically complex functions.

The three main goals of the TableGen programming language are:

- To provide quick and easy data entry
- To allow the user to refer to data by name
- To offer simple formulation techniques for generating algorithmically complex data

3.2 Lexical Conventions

3.2.1 Context-free Grammar Notation

Patterns of characters representing function in the TableGen language are described here using a context-free grammar notation where all valid TableGen programs can be produced by using the following rules. "Production rules" have a "non-terminal" on the left of the $\rightarrow$ symbol and a sequence of 'terminals', 'non-terminals', and added symbolic characters on the right side. All non-terminals are denoted with italics (ie. *nonTerminal*) whereas all terminals are denoted with fixed-width text and are surrounded by one quote character (ie. `'terminal'`):

$$foo \rightarrow \texttt{'bar'} \mid a$$

The following may appear as symbolic characters on the right side of a production rule denoting the specified meaning:

- () — placed around an arbitrary sequence of valid symbols for the right side of a production rule, they create a parenthetical group so that any of the following symbols operate on the entire group as in '$(a|b)*$'
- | — placed between two values (symbol or parenthetical group), it denotes an option between the two as in '$a|b$'
- ? — placed after a value it denotes that the value may appear either 0 or 1 times
- $+$ — placed after a value it denotes that the value appears 1 or more times
- $*$ — placed after a value it denotes that the value appears 0 or more times
- [value-value] — placed where a value would be placed and indicates a range of values

Juxtaposition denotes concatenation or the first value followed by the second value and requires parentheses to denote precedence if any of the above symbols are involved.

The following production rules are defined for use in later sections:

digit	$\rightarrow$	`'0'|'1'|'2'|'3'|'4'|'5'| '6'|'7'|'8'|'9'`
character	$\rightarrow$	any ASCII character
vartype	$\rightarrow$	`str` | `num`

3.2.2 Program Structure and Commenting

Unlike Java and C which use braces and parentheses to break code into separate segments, TableGen uses white space characters (tabs, newlines, carriage returns and spaces). As a result, TableGen requires much less punctuation than C-like languages.

Comments allow for annotations by the programmer and are ignored by the TableGen interpreter. There are two forms of comments: single-line and multi-line. Single-line comments begin with the two characters '`//`' and continue through the next newline character. Multi-line comments begin with '`/*`' and continue through the first subsequent appearance of the two characters '`*/`'.

3.2.3 Literals

Constant values for both types as defined in §3.3 can occur in a TableGen program.

1. Numbers: a number constant is a sequence of digits that optionally contains a decimal point and is optionally preceded by a '`-`' character to represent negative numerical values.
2. Strings: a string is a sequence of ASCII characters surrounded by double quotes ('`"`'). If a string is to contain any of the following characters, they must be preceded by a backslash ('`\`'):

   `"` `-` `+` `\` `,` `=`

3.2.4 Identifiers

General Identifiers

An identifier is a sequence of characters that represents the name of function or variable. It begins with either any character in the English alphabet (either lowercase or uppercase) or an underscore, followed by any number of alphanumeric characters or an underscore ('_').

Reserved Identifiers

All identifiers with names beginning with an underscore character ('_') are reserved for use by the interpreter—programmers may not use identifiers with a leading underscore as identifier names. Furthermore, the following identifiers are used as "keywords" and are reserved for controlling the language:

Types	Control Flow	Iteration	Functions

`str`
`num`
	`if`
	`elseif`
	`else`
		`while`
		`for`
		`foreach`
		`in`
			`func`
			`return`

Reserved keywords that are functions implemented by the TableGen interpreter:

- `_sum(var)` — used to calculate the sum of the elements in the passed variable
- `_avg(var)` — used calculate the average of the elements in the passed variable
- `_length(var)` — returns the number of elements in a variable
- `_tail(var)` — returns the last element in a variable

3.2.5 Declarations

No explicit declarations of variables or their types prior to their use is required for programming in TableGen. However, the first time a variable is used it may optionally be preceded by either '`num:`' or '`str:`' to denote that it may only hold either number or string values respectively. Functions are declared as valid for use as they are defined.

3.2.6 Scope

A block of code is a sequence of statements (see §3.5.2 on page 24) preceded by one extra level of tabular indentation than the surrounding code, or at no indentation in the base case and can only follow certain type of statements.

Code blocks denote a new scope that resides in the previous scope corresponding to the block in which the first block is contained. Variables can be referenced successfully in the scope in which they are first used or declared with type constraints as well as in all scopes contained therein.

As a special case, function parameters as well as variables declared or first used in the part of iterator and conditional statements preceding the code block are visible within the code block but not in the indentation scope in which they are first used or declared.

3.3 Types

TableGen has two data types:

- Numbers— any real number that can be represented by the primitive data types in the Java 1.5 programming language with defined operators for addition, subtraction, multiplication and division excluding ones expressed with exponential notation
- Strings — A string of characters

Where boolean options are required such as in conditional statements (§3.5.3), The value there must be a numeric type that is interpreted as follows:

- False if the value is zero
- True otherwise

3.4 Expressions and Operators

3.4.1 Primary Expressions

Function Calls (FuncCall)

This allows one to execute a subroutine by specifying the required parameters as per the function definition (§3.5.4, page 25). This is the only language construct that uses parentheses aside from parenthetical expressions, as it uses them to surround the function arguments. The arguments of a function call may have a name associated with it to match it up with the parameter of the same name in the function definition; either all arguments must be named, or all arguments must be unnamed.

Examples:

```
myFunc(var1 3 "asdf")
```

```
myFunc2(var1:var numParam:3 strParam:"asdf")
```

Hash Reference (HashRef)

Hash references allow one to access values in a variable list via its specified hash key by specifying the variable, followed by the dot operator, followed by the hash key.

Example:

```
varWithElements.hashKey
```

Parenthetical Expressions (ParExpression)

Simply a standard expression surrounded by parentheses that groups everything inside so that the inside is evaluated before any operations are performed for operators outside the parentheses.

Example:

```
(1 * var1) - funcCall(param1:value1)
```

Formal Primary Expression Grammar

The following rules describe all primary expressions

$$PrimaryExpression \rightarrow Identifier$$

$$
\begin{aligned}
&| \quad \mathit{Literal} \\
&| \quad \mathit{ParExpression} \\
&| \quad \mathit{FuncCall} \\
&| \quad \mathit{HashRef} \\
\mathit{Identifier} &\rightarrow (\texttt{`\_`} \mid \texttt{[a-z]} \mid \texttt{[A-Z]} \mid \mathit{digit})(\texttt{[a-z]} \mid \texttt{[A-Z]} \mid \mathit{digit})* \\
\mathit{HashRef} &\rightarrow \mathit{Identifier} \quad \texttt{`.`} \quad \mathit{Identifier} \\
\mathit{Literal} &\rightarrow \mathit{StringLiteral} \\
&| \quad \mathit{NumberLiteral} \\
\mathit{StringLiteral} &\rightarrow \texttt{`"`} \quad (\mathit{character})* \quad \texttt{`"`} \\
\mathit{NumberLiteral} &\rightarrow (\mathit{digit})+ \quad \texttt{`.`} \quad (\mathit{digit})+ \\
\mathit{ParExpression} &\rightarrow \texttt{`(`} \quad \mathit{Expression} \quad \texttt{`)`} \\
\mathit{FuncCall} &\rightarrow \mathit{Identifier} \quad \texttt{`(`} \quad \mathit{FuncParams} \quad \texttt{`)`} \\
\mathit{FuncParams} &\rightarrow (\mathit{vartype} \quad \texttt{`:`})? \quad \mathit{Identifier}
\end{aligned}
$$

3.4.2 Unary Expressions

There is only one unary expression—a dash before a digit denoting the negative of the digit's value:

$$\mathit{UnaryExpression} \rightarrow \texttt{`-`} \quad \mathit{Expression}$$

Examples:

```
-20
```

```
-a
```

3.4.3 Operators, Precedence and Full Expressions

All binary operators are left-associative.

Arithmetic Operators

(MultOp: `*` `/`, AddOp: `+` `-`) All arithmetic operators are binary and behave as standard arithmetic operators and are only defined for integer operands —see §3.6 on page 29 for information on type conversions

Example:

```
a + 3 - b + 2
```

Boolean Comparison Operators

(BoolComp: < <= > >= == !=) 'less than', 'less than or equal to', 'greater than', 'greater than or equal to', 'equal', and 'not equal' respectively. They are all binary operators that evaluate to '0' if the corresponding (in)equality is false and a non-zero value otherwise. All inequality operators (< <= > >=) are only defined for numerical operands while the two equality testing operators (== !=) are defined for numerical types as well as strings as long as both operands are of the same type. If operating on two string values, the equality operator (==) will evaluate to true if the two strings are identical and false otherwise, and the inequality operator will do the opposite; string comparison is done using the methods in the Java 1.5 programming language.

Example:

```
a * 3 * b / random_variable
```

Logical Operators

(LogicalOp: && ||) 'logical and' and 'logical or' operators respectively. && evaluates to 0 if either of the operands is zero or a non-zero value otherwise, || evaluates to 0 if both of the operands are zero or a non-zero value otherwise. Logical operators are only defined for numerical types.

Example:

```
is_known || is_partially_known && 1
```

Assignment/Put Operators

(AssPutOps: ->, =) The put-op operator (->) takes the expression on the left and places it into the coordinate on the right. The assignment operator takes the value of the expression on the right and stores it to the location on the left. Neither of these operators can contain another assignment or put operator in their sub-expressions.

Precedence Hierarchy and Grammar Rules

The setup of the following grammatical rules specifies the levels of precedence in the table below:

highest	.	
	,	(StrConOp)
	* /	(MultOp)
	+ -	(AddOp)
	< <= > >= == !=	(BoolComp)
	&& , ||	(LogicalOp)
lowest	-> =	(AssPutOps)

$$
\begin{aligned}
\mathit{Expression} &\rightarrow \mathit{LogicalExpr}\ ((\mathit{AssPutOps}\ (\mathit{LogicalExpr} \mid \mathit{HashRef}))\\
&\qquad \mid (\mathit{AssPutOps}\ \ \mathit{Coord}))*\\
\mathit{LogicalExpr} &\rightarrow \mathit{BoolExpr}\ \ (\mathit{LogicalOp}\ \ \mathit{BoolExpr})*\\
\mathit{BoolExpr} &\rightarrow \mathit{AddExpr}\ \ (\mathit{BoolOp}\ \ \mathit{AddExpr})*\\
\mathit{AddExpr} &\rightarrow \mathit{MultExpr}\ \ (\mathit{AddOp}\ \ \mathit{MultExpr})*\\
\mathit{MultExpr} &\rightarrow \mathit{StrConExpr}\ \ (\mathit{MultOp}\ \ \mathit{StrConExpr})*\\
\mathit{StrConExpr} &\rightarrow \mathit{PrimaryExpression}\ \ (\mathit{StrConOp}\ \ \mathit{PrimaryExpression})*\\
\mathit{Coord} &\rightarrow \text{`[`}\ \ \mathit{LogicalExpr}\ \mid\ \mathit{LogicalExpr}\ \ \text{`]`}
\end{aligned}
$$

3.5 Statements

3.5.1 Normal Statements

A normal statement is simply an expression followed by an end-of-line character.

3.5.2 Block Statement

Block statements are one or more normal statements that are indented one tab level from the surrounding code:

$$\textit{BlockStmt} \rightarrow \texttt{tab}\ \ \textit{NormalStmt}+\ \ \texttt{tab}$$

3.5.3 Conditional Statements

Conditional statements are composed of an 'if statement' optionally followed by one or more 'elseif' statements optionally followed by an 'else' statement where each component is defined in the following subsections.

If Statement

The `if` keyword followed by a condition expression that must evaluate to a numeric type for interpretation as a boolean value (§3.3) followed by a block of code to execute if the condition expression evaluates to true.

Example:

```
if a <= 20
    a = a + 4
    a -> 0 , funcCall(name:"asdf")
```

Elseif Statement

The `elseif` keyword followed by a condition expression that must evaluate to a numeric type for interpretation as a boolean value (§3.3) followed by a block of code to execute if both the condition expression evaluates to true and the condition expressions of all preceding if/elseif statements were false when evaluated.

Example:

```
elseif b != 4
    drawCircle(radius:3 centerX:2 centerY:b)
```

Else Statement

The `else` keyword followed by a block of code to execute if the condition expressions of all preceding if/elseif statements were false when evaluated.

Example:

```
else
    "else statement" -> _row_head.theRow , _col_head.theCol
```

Putting It All Together (an Example)

```
if testVar < 10 && testVar > 0
    value = value + 1
elseif testVar < 20
    value = value + 10
else
    error = "invalid value for testVar"
```

3.5.4 Function Definitions

Function definitions begin with the `func` keyword optionally followed by a list of function parameters (each of which have a name and may or may not have a specified type restriction) followed by a block of code to execute when the function is called with valid parameters.

Example:

```
func myFunction str:name num:id random_data
    name -> id , 0
    random_data -> id , 1
    myOtherFunction(id:id+3 caller:"myFunction")
```

3.5.5 Return Statement

The return statement may only appear inside a function definition block or any blocks inside the block and consists of the `return` keyword followed by an arbitrary expression. When a return

statement is encountered, execution of the function halts and the return value— the value of the expression following the `return` keyword—is made available to the context from which the function call was made.

Example:

```
return (numerator / denominator - 23) * funcCall(a)
```

3.5.6 Iterator Statements

The following statement types are used for iterative execution of code blocks.

While Statement

The while statement is the `while` keyword followed by a condition expression that must evaluate to a numeric type for interpretation as a boolean value (§3.3) followed by a block of code to execute if the condition expression evaluates to true.

Example:

```
while a <= 10
    doFunctionCall(first_param:my_var)
    my_var * 3 - 2 -> a , 0
```

For Statement

The for statement is similar to the while statement, except for the insertion of two expressions. The syntax of the for loop are the `for` keyword followed by an expression followed by a semicolon (`';'`) followed by a condition expression followed by a semicolon followed by an expression followed by a code block.

The first expression is executed prior to checking the condition expression, and the final expression is executed immediately prior to each subsequent check of the condition expression. The condition expression must evaluate to a numeric type for interpretation as a boolean value (§3.3), and the trailing code block will execute only if the condition expression evaluates to true, and will continue the pattern of testing the condition expression's value and executing the code until the condition expression evaluates to false.

Example:

```
for num:i = 0 ; i < _length(list_var) ; i = i + 1
    a -> i,1
    a = a + b
```

Foreach Statement

The foreach statement is a construct for iterating through all values in a given variable. Its syntax is the `foreach` keyword followed by an identifier followed by the `in` keyword followed by an identifier followed by a code block.

The first identifier is a variable that may be declared as restricted to a type and is automatically assigned the value of the next element in the variable specified by the second identifier. The block of code will execute once for each element in the variable specified by the second identifier until every item after each given iterator position has caused the code block to be executed.

Example:

```
foreach row in _row_header
    if row / 2 == 0
        "even row" -> row , 0
```

3.5.7 Formal Statement Grammar

$$
\begin{aligned}
\mathit{Statement} &\rightarrow \mathit{NormalStmt} \\
&\quad | \ \mathit{CondStmts} \\
&\quad | \ \mathit{FuncDef} \\
&\quad | \ \mathit{ItrStmts} \\
\mathit{NormalStmt} &\rightarrow \mathit{Expression} \quad \mathit{EOLC} \\
\mathit{EOLC} &\rightarrow \texttt{newline} \\
&\quad | \ \texttt{carriageReturn} \\
&\quad | \ \texttt{newline carriageReturn} \\
\mathit{CondStmts} &\rightarrow \mathit{IfStmt} \quad \mathit{ElseIfStmt}* \quad \mathit{ElseStmt}? \\
\mathit{IfStmt} &\rightarrow \texttt{if} \quad \mathit{Expression} \quad \mathit{BlockStmt} \\
\mathit{ElseIfStmt} &\rightarrow \texttt{elseif} \quad \mathit{Expression} \quad \mathit{BlockStmt} \\
\mathit{ElseStmt} &\rightarrow \texttt{else} \quad \mathit{BlockStmt} \\
\mathit{FuncDef} &\rightarrow \texttt{func} \quad \mathit{FuncParam}* \quad \mathit{BlockStmt} \\
\mathit{FuncParam} &\rightarrow (\mathit{Vartype} \quad \texttt{':'})? \quad \mathit{Identifier} \\
\mathit{ReturnStmt} &\rightarrow \texttt{return} \quad \mathit{Expression}
\end{aligned}
$$

$$
\begin{aligned}
\textit{ItrStmts} &\rightarrow \textit{WhileStmt} \\
&\mid \textit{ForStmt} \\
&\mid \textit{ForeachStmt} \\
\textit{WhileStmt} &\rightarrow \texttt{while}\ \textit{Expression}\ \textit{BlockStmt} \\
\textit{ForStmt} &\rightarrow \texttt{for}\ \textit{Expression}\ \texttt{`;`}\ \textit{Expression}\ \texttt{`;`}\ \textit{Expression}\ \textit{BlockStmt} \\
\textit{ForeachStmt} &\rightarrow \texttt{foreach}\ \textit{Indentifier}\ \texttt{in}\ \textit{Identifier}\ \textit{BlockStmt}
\end{aligned}
$$

3.6 Automatic Type Conversion

While the TableGen programming language provides a uniform interface for all real numbers, because of the hardware on which the TableGen interpreter is designed to be implemented on there must be a distinct difference between integer values and non-integer values. This leads to the following convention for automatic variable type conversion where integers and floating point/double-precision floating point numbers are as specified in the Java 1.5 programming language:

> For any arithmetic operator (+ - * /), If both operands are integers the result is an integer—otherwise the result is a double-precision floating point number

Chapter 4

Project Plan

4.1 Process Used For Planning, Specification, Development, And Testing

4.1.1 Planning

After midterms, Del made an individual effort to create an outline for the Symbol Table and intermediate Table Generation code. He communicated his outline with the rest of the team. Daniel expanded the outline and wrote an initial draft of a portion of the symbol table. Del re-wrote Daniel's symbol table implementation because it was not well commented (it also was not fully integrated with generated grammar code). Del proceeded to complete a working draft of the table generator. Daniel picked up from the there by debugging, adding to, and finishing the table generator. Andrey communicated to the team how output should be formatted for the postprocessors. Del communicated to Andrey what the the preprocessor should feed the grammar.

4.1.2 Specification

After midterms, Del made an individual effort to create an outline for the Symbol Table and intermediate Table Generation code. He communicated his outline with the rest of the team. Daniel was able to use Del's outline to create a large portion of the symbol table. Del re-wrote Daniel's abstract symbol table because it was not well commented. Del proceeded to complete the table generator. Daniel picked up from the there by finishing the table generator. Andrey communicated to the team how output should be formatted for the postprocessors. Del communicated to Andrey what the the preprocessor should feed the grammar.

4.1.3 Development

Development took place at all instances when group members had time to work on the project. We were never waiting for one member to finish something before starting on our respective parts. This is mostly due to the fact that we were very clear about the specifications. Each of us knew what we were going to get as input and what we had to output. Group members were able to create their own "stress test" files and debug their code with them.

4.1.4 Testing

Andrey used his testing infrastructure to give feedback to the team as well as himself about what worked in the code and what did not. Andrey's testing infrastructure would test TableGen through all levels of intermediate output and final output. Team members tested their code carefully before committing by creating their own samples of intermediate output or using Andrey's test scripts.

4.2 Programming Style Guide

We used the following lists of guidelines:

4.2.1 General

- If you need to use an non-obvious trick comment it well.
- Split very long sections of code into subroutines. Try to keep length of code block less than 100 lines.
- Variable names shouldn't be too short, nor should they be too long.
- Use uppercase letters or underscores for multi-word variables.
- Split lines obvious by indenting the over-flowed line.

4.2.2 Perl

- Make that all lines are less than 125 characters long.
- Do not take advantage of Perl tricks and short cuts that make code hard to read.
- When printing code with backslashes ("\") use single quotes.

- When printing html use qq ().
- If you pass arrays as arguments to a subroutine, use references — unless there is a reason not to.
- Variable names shouldn't be too short, nor should they be too long.
- Make use of die when opening file handles and other external things that might cause an error.
- Use "strict" and turn warnings on.
- Add all global variables to `use var qw [ ... ]`.

4.2.3 Java

- Make that all lines are less than 125 characters long.
- Capitalize constant variables (final).
- Use assert often.
- Use javadoc conventions when commenting on public classes.

4.3 Timeline

Date	Documentation	Main Executable	Grammar	Symbol Table	Table Generator	Testing
2007-09-13	Proposal: Start					
2007-09-20	Proposal: Draft One					
2007-09-22	Proposal: Draft Two					
2007-09-22	Proposal: Final Draft					
2007-09-28						Test Script: Start
2007-10-04			Start			
2007-10-07	LRM: Start		Grammar: First Draft			
2007-10-12	LRM: Draft One					First Sample

2007-10-16		Pre- processor: Draft One				A Few More Samples
2007-10-18	LRM: Final Draft	Second Draft				
2007-10-24			Working Draft			
2007-11-10		Pre- processor: Draft Two				Test Script: Draft One
2007-11-13		Main Exec: Draft One		Skeleton		
2007-11-20				Draft Two		
2007-11-25		Plain Text Output			First Draft	
2007-11-30					Second Draft	
2007-12-10	Final Report: Start	LaTeX & HTML Output	Working Draft	Working Draft	Second Draft	
2007-12-12	Final Report: First Draft					Fully automated and working
2007-12-16				Complete	Complete	
2007-12-18	Final Report Complete					

4.4 Roles and Responsibilities

Name	Role	Responsibilities
Andrey Falko	Team Leader	Preprocessor, Postprocessors, Test Scripts and Testing, Build Scripts, Drafting Documentation
Del Slane	Second-in-Command	Front-end, Relevant Documentation
Daniel Pestov	Contributor	Backend: Symbol Table, Relevant Documentation
Timothy Washington	Contributor	Backend: Table Generator, Relevant Documentation

4.5 Software Development Environment

Used Andrey's dorm machine to host our code and to optionally develop on. Andrey's machine runs Gentoo Linux. Andrey gave shell access to all members of the group. Andrey also created a shared http account. We hosted our code in a Subversion repository on Andrey's machine. Subversion was configured for use with the svn+ssh protocol as well as http. Members of the group had the following choices for using Subversion:

1. Use Eclipse with the subclipse module where members would configure it to do operations via http or svn+ssh.
2. Use command line from a remote Mac OS machine or Linux machine via svn+ssh or http.
3. Login to Andrey's server via ssh and get copies of the code locally.

Andrey's machine provided all of the tools needed to build and run TableGen. Andrey's machine also included command line text editors such as vim and nano (a open sourced pico re-write). Other group members' machines needed to have, at the very minimum: Java and ANTLR. If group members wanted to run test scripts and the main executable, their machines needed to have Perl with one package from cpan and make, in addition to Java and ANTLR. Those members who ended up working on their own machines used Eclipse, where they added ANTLR to the class path. Those users who wanted to work on documentation needed to have LaTeX installed.

At one time we created a branch for Eclipse users. Eclipse users placed their Eclipse environment under version control. This enabled them to be more productive. Andrey, the only full-time command-line user, created scripts that allowed him to synchronize the branch to trunk, there a command line build and test environment resided.

Here is a summary from our environment:

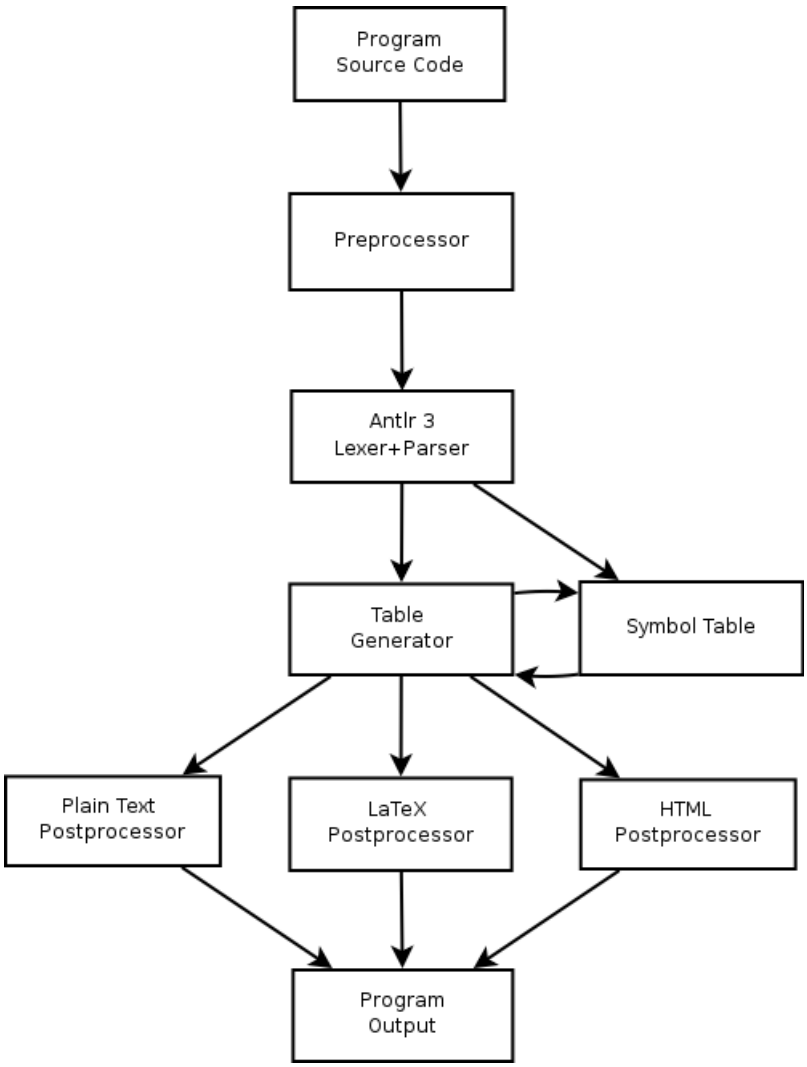
Version Control System	subverison-1.3.2
Operating Systems Used	Windows, Max OS X, GNU/Linux
Development Environments	Eclipse, Netbeans, Bash shell
ANTLR	antlr-3.0.1
Java	sun-jdk-1.5.0.13
Perl	perl-5.8.8
Make	make-3.81

4.6 Project Log

Our project log is generated using `svn log`. Note that those members who chose to use svn with http, show up as "4115" instead of their actual name. This is because the http user they were connecting through was called 4115. See the log in our Appendix: §8.6

Chapter 5

Architectural Design



5.1 Description of Components

In our diagram above, a TableGen program abiding by our LRM, is first fed into the preprocessor. The preprocessor adds open and closed braces for every block of code and formats coordinates so that the grammar can parse them without ambiguity. The preprocessor, which is part of our main executable creates a temporary file that gets fed into an ANTLR generated lexer and parser which creates an Abstract Syntax Tree.

We walk the tree into two places: The Symbol Table and the Table Generator (§8.1.4). The Symbol Table creates lists of symbols for every scope in the TableGen program. The Table Generator walks the Abstract Syntax Tree and uses the symbol table in order to resolve variables as well as types. The Table Generator parses all of the source program's statements and based on them, creates a two dimensional vector that is outputted at the end.

The output that gets fed into the postprocessors is tab and newline separated text. Everything on the same row gets separated by tabs, while new rows are separated by newlines. The user, through the main executable, chooses what postprocessor will be used. The postprocessor will traverse the tab and newline separated text and produce formatted text.

5.2 Who Implemented What

Andrey implemented both the preprocessor and all of the postprocessors, and made a few edits to the grammar and interpreter. Andrey also completed his testing responsibilities (see §6.3).

Del implemented most of the ANTLR grammar, most of the symbol table, and part of the interpreter.

Daniel implemented some of the Symbol Table and implemented parts of the interpreter.

Tim made an initial version of the interpreter's table printing method.

The final report was largely written by Andrey, and Del did the re-write of the LRM prior to submission to correct feature specification and syntax conventions as well as provide a more formal description of the language's grammar.

Other documentation is outlined by Andrey, partially drafted by Andrey, and completed by Daniel and Del.

Chapter 6

Test Plan

Our test plan to was to test every state of output of our Interpreter. The first stage was to test the preprocessor. The second stage was the unformatted table. The final stage consisted of testing the output of each post-processor.

6.1 Test Samples

6.1.1 Pascal Example

```
/* Factorial algorithm using a cache to compute factorials. */
num:factCache = 1
func factorial num:n num:factCache
        if n < _length (factCache)
                return factCache[n]
        prod = _tail (factCache)
        while (_length (factCache) <= n)
                /* Append prod onto end of factCache array. */
                prod = prod * _length (factCache)
                factCache = factCache, prod                                    10
        return prod

func pascalCombination num:n num:k
        return factorial (n factCache) / (factorial (k factCache) * factorial ((n−k) factCache))

max = 10 /* Set maximum # of rows to output */

i = 0
j = 0
while (i < max)                                                                20
        while (j <= i)
                pascalCombination (i j) −> i, j
                j = j + 1
```

```
        i = i + 1
        j = 0
```

Plain Text Output:

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84 36 9   1
```

6.1.2 New Jersey Nets Example

```
// This program returns statistics regarding how well players on the
// New Jersey Nets play with each other.

// Negative number means that the player missed the game.
Kidd = 7, 2, 16, 9, 6, 17, 14, 19, 11, 16, 2, 12, 14, 15, 12, 19, 6, 7, −1, 10
Carter= 24, 7, 18, 12, 24, 17, −1, −1, −1, −1, −1, 13, 15, 19, 32, 24, 22, 19, 19, 32
Jefferson = 29, 27, 22, 25, 25, 28, 32, 21, 15, 18, 22, 30, 30, 27, 27, 23, 21, 36, 31, 29

str:players = "Jason Kidd", "Vince Carter", "Richard Jefferson"

num:row = 0
num:col = 1
foreach player in players
        player −> row,col
        player −> col,row
        col = col + 1

printPPG (Kidd 1)
printPPG (Carter 2)
printPPG (Jefferson 3)

compare (player1:Kidd coord1:1 player2:Carter coord2:2)
compare (coord2:3 player1:Kidd player2:Jefferson coord1:1)
compare (Carter 2 Kidd 1)
compare (Carter 2 Jefferson 3)
compare (Jefferson 3 Kidd 1)
compare (Jefferson 3 Carter 2)

func printPPG player coord
        num:totalPoints = 0
        num:gamesPlayed = 0
```

```
        // Find per game average
        foreach score in player
                if (score >= 0)
                        totalPoints = totalPoints + score
                        gamesPlayed = gamesPlayed + 1

        // Round to the nearest tenth.
        _round (totalPoints / gamesPlayed 10.0) -> coord, coord

func compare player1 coord1 player2 coord2
        num:totalPoints = 0
        num:gamesPlayed = 0

        // If both played
        for i = 0; i < _length (player1); i = i + 1
                if player1[i] >= 0 && player2[i] >= 0
                        totalPoints = totalPoints + player1[i]
                        gamesPlayed = gamesPlayed + 1

        _round (totalPoints / gamesPlayed) -> coord1,coord2
```

Plain Text Output:

	Jason Kidd	Vince Carter	Richard Jefferson
Jason Kidd	11.3	10.86	11.26
Vince Carter	19.86	19.8	19.8
Richard Jefferson	25.63	27.33	25.9

6.1.3 Hash Keys Example

```
people = American>>"Betty", Russian>>"Vladimir", German>>"Konstantine"
printTheGerman (people)

func printTheGerman lister
        lister.German -> 0,0
```

Plain Text Output:

```
Konstantine
```

6.1.4 Test Suite

We used one master script to test our Interpreter. Let's look at the script section-by-section (see appendix for the code §8.5.1):

In lines 10-20 we initialize some global variables to be used by subroutines later on. After that we call five tests. The first tests the preprocessor, the second tests the intermediate output, and the others test the different post-processors.

All of the subroutines that follow essentially do the same thing. They call our main TableGen executable with an option requesting a type of output. They eat the output that TableGen provides and call our `diff` function which compares the sample out to desired output.

The `printReal` subroutine (lines 70-76) writes to hard disk the output of TableGen for the given sample. The `diff` subroutine (lines 77-108) uses the GNU/Linux tool diff in order to compare the output of the interpreter to the desired output. If diff does not output anything, then there the outputs are the same and the test passes, otherwise, our script prints the details of what failed such as the outputs and the stage where failure occurred.

You can see more example in the appendix: §**??**

6.2 Test Automation

We used a Makefile for automation along with a script that automatically finds new samples and adds appropriate targets to the Makefile. In order to test all cases, people needed to run:

```
make test
```

In order to run an individual test people would needed to run:

```
make <sample name>
```

In order to add a test, a person would have to create the program they want to test in the `samples` directory, in additional to their desired output. While inside of TableGen's main directory, the person would have to run:

```
test/add-prog.pl
```

This would add a target to the Makefile and modify the general `test` target.

6.3 Who Did What

Andrey wrote all of the testing scripts and created all of the test samples. Andrey also conducted most of the testing and provided feedback to everyone via phone or email.

Chapter 7

Lessons Learned

7.1 Andrey Falko

I take away from this project three categories of lessons: leadership, time management, and code organization. I was assigned the task of being team leader. I learned that a team leader has to allocate work according who is best at what. To do this, the team leader needs to talk one-on-one with each member and find out what projects they worked on before. By doing this, the leader places people in positions where they are most comfortable and where they will do a better job. The leader's job should not be to tell everyone how to do every detail, rather the leader needs to defer work to those that can do it better. Another thing that a leader need to do well is manage discontent. There times when team-mates will be angry with each other. The leader needs to reason with everyone individually and calm them down. The leader should expect arguments to arise regarding implementation that are like the chicken and the egg problem. In these situation the leader needs to clamp down make the decision.

We started early, but did not continue our early start. This was partly due to midterms and other school work, but also due to the fact that we felt we needed large blocks of time to be productive. I think this is the wrong way to think about this project. If you spend two hours every Sunday and Saturday during your "busy weeks", you won't hurt your study time for other classes. In fact, I think you should spend time on your project in between studying as a way of "taking a break" because usually a person gets tired because they are doing something tedious and simply need to put their mind on something else.

Make sure that you organize all of your code as early as possible. This organization will allow you to reduce stress by creating certainty. If you and your team-mates know what they need to code and roughly how long it will take them, you will be happy throughout the entire semester.

One last thing that I learned was that my previous experience in trying to contribute to various open source software projects was extremely valuable for this project. I was able to get into our code fast enough to fix a number of pesky last-minute bugs. I therefore urge all aspiring computer

scientists to make attempts to contribute to open source software projects. You are not only helping the global community by doing so, you are mostly helping yourself become a better programmer within a group.

7.2 Daniel Pestov

The most valuable lessons I learned while doing this project are: 1) comment your code well and 2) use functions as much as possible and as early as possible to provide a greater degree of modularity and make your code more readable and easy to understand. This is the first I've worked in a group on a large software development project and my coding ethics had to change in order to reflect that. While I always submit clean and consistent code and use comments as much as possible, I usually perform these revisions at the very end. Working in a team taught me the importance of early commenting and making sure that even non-final code needs to be clean and understandable because when your teammates spend extra time trying to understand or debug your code, the productivity of the entire group is substantially minimized.

7.3 Del Slane

I have learned the fundamental issues with compiler design that I could not have without a hands-on experience, although it's almost implied. The other significant area of learning has been in assessing the work habits of others.

7.4 Timothy Washington

I have learned a few valuable lessons as a result of this project. First and foremost, there are a lot of improvements I need to make as far as the way I work with groups, especially when it comes to programming assignments. Prior to this project I had a good deal of programming experience, designing and building programs as large as 8,000 lines. The problem was that I always did these things by myself, never collaborating with others. Looking at thousands of lines of code and understanding how it works is easy to do when you're the one who wrote it. However, coming into a project, a great deal of which has been done, and trying do add new features and debug it can be very complicated; especially with something as elaborate as a compiler.

In my future group assignments (involving programming), I will be more ambitious as far as participating early on in the creation of programs as it will allow me to build something from the ground up rather than build on top of something that I may not fully understand the functionality of. The early lack of ambition in this case was caused by two factors. First of all I didn't get Internet access on my computer until midterm week which severely limited the extent to which I could work on

this project until then. The other reason was due to constant assignments from other classes. When one has a solo project due in three days and group project due in three months with group members eagerly undertaking task after task for it, the three day project will win out every time (which leads to another lesson to be learned, that being time management). Ultimately, when I was called on by my group members to implement certain features, I had every intention of making an exceptional contribution, but was unable to accomplish everything that I had planned to do.

Chapter 8

Appendix

8.1 Main Code

8.1.1 TableGen

This is our main executable.
Signed: Andrey Falko

```
#!/usr/bin/perl
#
# This will be the main TableGen executable and preprocessor.
#
use strict;

use vars qw [ $tgexec $debug $tmpExt $plain $outFile $onlyPre $noPost $scopeStarts
              $latex $latexVerLines $latexHorLines $latexHeader
              $html $htmlBorder $htmlHeader ];
$debug = 0;                                                                    10
$outFile = 0;
$onlyPre = 0;
$noPost = 0;
$plain = 0;
$html = 0;
$htmlBorder = 0;
$htmlHeader = 0;
$latex = 0;
$latexVerLines = 0;
$latexHorLines = 0;                                                            20
$latexHeader = 0;

$scopeStarts = "^((if)|(else)|(while)|(for)|(foreach)|(func))";

# Location of interpretor
```

```
$tgexec = 'main/TableGen';

# Temporary files extension
$tmpExt = ".tmpout";
                                                                                    30
my $prog = pop @ARGV;

my $index = 0;
my $skip = 0;
for (@ARGV) {
        $index += 1;
        if ($skip) {
                $skip = 0;
                next;
        }                                                                           40

        /^--help$/ && do { &help; next };
        /^--debug$/ && do { $debug = 1; next };
        /^--html-border$/ && do { $html@Border = &getNext ($index);
                                  &help ("html-border parameter is wrong. Must be integer.")
                                        if (!($html@Border =~ /^\d+$/));
                                  next };
        /^--html-header$/ && do { $html@Header = 1; next };
        /^--only-intermediate$/ && do { $noPost = 1; next };
        /^--only-preprocess$/ && do { $on@yPre = 1; next };                         50
        /^--output$/ && do { $outFi@e = &getNext ($index); next };
        /^--output-type$/ && do { my $type = &getNext ($index);
                          if ($type eq 'plain') {
                              $p@ain = 1;
                          } elsif ($type eq 'html') {
                              $html@ = 1;
                          } elsif ($type eq 'latex') {
                              $@atex = 1;
                          } else {
                              &help ("Error. Unknown type of output: $type.");      60
                          }
                          next };
        /^--latex-all-lines$/ && do { $@atexVerLines = 1; $@atexHorLines = 1; next };
        /^--latex-only-hor-lines$/ && do { $@atexHorLines = 1; $@atexVerLines = 0; next };
        /^--latex-only-vert-lines$/ && do { $@atexVerLines = 1; $@atexHorLines = 0; next };
        /^--latex-header$/ && do { $@atexHeader = 1; next };
        # More commandline options are to come. Follow
        &help ("Can't process request because I do not know what $_ means. \n");

        # Nested subroutine so that we keep the same scope.                         70
        sub getNext {
                my $index = shift;
                $skip = 1;
                return $ARGV[$index];
        }
}

sub help {
        my $error = shift;
```

```
        print STDERR "$error\n";
        print "TableGen expects the filename at the end of any commandline sequence: \n".
              "TableGen <options> [filename].\n\n";
        print "Available options: \n";

        print "\t--debug\t\t\t\t\tEnables debugging output and activities.\n";

        print "\t--help\t\t\t\t\tPrints this help menu.\n";

        print "\t--html-border [size]\t\t\t\tSets border of html chart to \"size\". Default 0.\n";
        print "\t--html-header\t\t\t\tPrint html title, body, and other common lines for a new html file.\n";

        print "\t--only-intermediate\t\t\t\tKeeps intermediate output of program without postprocessing.\n";
        print "\t--only-preprocess\t\t\t\tOutputs preprocessed code, but without running code.\n";

        print "\t--output [filename]\t\t\t\tPuts program output into filename.\n";

        print "\t--output-type [type of output]\t\tControls the type of output interpretor generates.\n";
        print "\t\tAvailable output types:\n";
        print "\t\t\thtml\n";
        print "\t\t\tplain\n";
        print "\t\t\tlatex\n";

        print "\t--latex-all-lines\t\t\t\tCreates horizontal and vertical lines for latex charts.\n";
        print "\t--latex-only-hor-lines\t\t\tCreates only horizontal lines for latex.\n";
        print "\t--latex-only-vert-lines\t\t\tCreates only vertical lines for latex.\n";
        print "\t--latex-header\t\t\t\tPrint latex being and end statement for a new latex file.\n";
        # Describe any options that you add here. New Print statement for each option.
        exit 0;
}

# Make sure that we output something.
if ($html == 0 && $latex == 0) {
        $plain = 1;
}

# Check input:
if (! -e $prog) {
        &help ("Error: The filename ($prog) of the program you wish to run does not exist.\n");
}

# Take program and shove it into memory:
open PROG, $prog or die "Could not open $prog: $!\n";

my @lines = <PROG>;

push @lines, "\n"; # Last element needs to end with a newline for the preprocessor.

# Run proprocess. Pass it as reference because we do not want
# to make copies of a potentially big list of lines.
my $code = "";
&preprocess (\@lines);

if ($outFile) {
```

```
        open OUT, $outFiℓe or die "Can't open $outFile for output: $!\n";
        select OUT;
}


# Exit if user only wants to see preprocessed output.
if ($onℓyPre) {                                                          140
        print $code;
        exit 0;
}

# Now take the pre-processed code and run it through the interpreter:
my $out = &getTmpOut ($prog);
open TMP, ">$out" or die "Can't write to temporary file: $!\n";
print TMP "$code";
my $basicOutput = `java $tgexec $out`;
unlink "$out" if (!$debug);                                              150

if ($noPost) {
        print $basicOutput;
        exit 0;
}



# Debugging framwork below.
                                                                         160
#open TMP, $prog;

#my $basicOutput = "";
#$basicOutput = $basicOutput . $_ for (<TMP>);

# Debugging framework end.



&printPlain ($basicOutput) if ($pℓain);                                  170
&printHtml ($basicOutput) if ($htmℓ);
&printTex ($basicOutput) if ($ℓatex);

# Subroutines below...
sub preprocess {
        my $ref = shift;

        my $wscnt = 0;
        my $brak = 0;
        my $comment = 0;                                                 180
        for my $next (@{$ref}) {
                # Don't mess with comments
                $comment = 1 if ($next =~ /\/\*/); # /*
                $comment = 0 if ($next =~ /\*\//); # */
                if ($comment) {
                        $code = $code . $next;
                        next;
```

```
}

# Remove new lines at end...we add them later.                          190
next if (/^\n$/); # Skip all new lines.
chomp ($next);

#Place square brackets around coordinates and make separator
#a pipe
$next =~ s/->\s*(.+\s*),(\s*.+)\s*$/-> \[$1\|$2\]/;

#Brackets at begining of tab increase.
if ($next =~ /^(\t+)/) {
        my @ws;                                                         200
        if (!$1) {
                die "This should never happen.";
        } else {
                @ws = split //, $1;
        }

        if (@ws > $wscnt) {
                $next =~ s/^/$1 \{\n/;
                $brak += 1;
        } elsif ($brak > @ws) {                                         210
                while ($brak > @ws) {
                        my $tabs = "\t" x ($brak - 1);
                        $next =~ s/^(\t+)/\n$tabs\}\n$1/;
                        $brak -= 1;
                }
        } elsif (@ws < $wscnt) {
                $next =~ s/^(\t+)/\n$1\}\n$1/;
                $brak -= 1;
        } else {
                # Add the new line back.                                220
                $code = $code . "\n";
        }
        $wscnt = scalar @ws;
        $code = $code . "$next";

        next;
} elsif ($brak > 0) {
        # Here we "abruptly" exit scope, and need to fill out all
        # the brakets we need. The number of brakets is
        # coincidentally the number of tabs.                            230
        while ($brak > 0) {
                $code = $code . "\n" . ("\t" x ($brak - 1));
                $code = $code . "}\n";
                $brak -= 1;
        }
        $wscnt = 0;
} else {
        $code = $code . "$next";
        # This is keep open brakets on same line as start of
        # scope.                                                        240
        $code = $code . "\n" if (!($next =~ /$scopeStarts/));
```

```
                }

        }
}

sub getTmpOut {
        # Take only the last part of path. And append
        # a tmp file extension.
        my $path = shift;

        if ($path =~ /\/(.+)$/) {
                return $1 . $tmpExt;
        } else {
                return $path . $tmpExt;
        }
}

sub printPlain {
        my $out = shift;
        my @rows = split /\n/, $out;

        my $maxW = &getMaxColWidth ($out) + 1;

        # Easy case first.
        for (@rows) {
                chomp;
                my @cols = split /\t/, $_;
                for (@cols) {
                        print $_;
                        # Leave the needed amount of space between column.
                        my @chars = split //, $_;
                        print " " x ($maxW - int (@chars));
                }
                print "\n";
        }
}

sub printHtml {
        my $out = shift;

        if ($htmlHeader) {
                print qq(<html>\n);
                print qq(<head>\n);
                print qq(<title>Title</title>\n);
                print qq(</head>\n);
                print qq(<body>\n);
        }

        print qq(<table border=$htmlBorder>\n);
        my @rows = split /\n/, $out;
        for (@rows) {
                print qq(<tr>\n);
                my @cols = split /\t/, $_;
                for (@cols) {
```

```
                print qq(<td>$_</td>\n);
            }
            print qq(</tr>\n);
        }

        if ($htmlHeader) {
            print qq(</body>\n);
            print qq(</html>\n);
        }

}

sub printTex {
        my $out = shift;

        if ($latexHeader) {
            # Give users good starting point:
            print '\documentclass[12pt]{article}' . "\n";
            print '\usepackage{pslatex}' . "\n";
            print '\usepackage{geometry}' . "\n";
            print '\geometry{verbose,letterpaper,tmargin=1in,bmargin=1in,lmargin=1.5in,rmargin=1in}' . "\n";

            print '\begin{document}' . "\n";

            print '\title{}' . "\n";
            print '\date{}' . "\n";
            print '\author{}' . "\n";

            print '\maketitle' . "\n";
        }

        print '\begin{tabular}{';

        my $colcnt = &numOfCols ($out);

        if ($latexVerLines) {
            print "|";
            print "c|" x $colcnt;
        } else {
            print "c" x $colcnt;
        }
        print "}\n";

        my @rows = split /\n/, $out;
        for (@rows) {
            my @cols = split /\t/, $_;
            print "\\hline\n" if ($latexHorLines);

            my $last = pop @cols; # Last element is special.

            for (@cols) {
                # Latex does not like these chars:
                s/&/\\&/g;
                s/%/\\%/g;
```

```
                s/\$/\\\$/g;

                print "$_ & ";
            }

            print $ℓast . " \\\\\n";
        }

        print "\\hline\n" if ($ℓatexHorLines);

        print '\end{tabular}' . "\n";

        if ($ℓatexHeader) {
            print '\end{document}' . "\n";
        }

}

sub numOfCols {
        my $out = shift;

        my @rows = split /\n/, $out;

        my $most = 0;
        for (@rows) {
            my @cols = split /\t/, $_;
            $most = int (@cols) if ($most < int (@cols));
        }

        return $most;
}

sub getMaxColWidth {
        my $out = shift;

        my @rows = split /\n/, $out;

        my $widest = 0;
        for (@rows) {
            my @cols = split /\t/, $_;

            for (@cols) {
                my @count = split //, $_;
                $widest = int (@count) if ($widest < int (@count));
            }
        }

        return $widest;
}

sub numOfRows {
        my $out = shift;

        my @rows = split /\n/, $out;
```

```
        return int (@rows);
}
```

8.1.2 gram/TableGen.g

This is our Antlr grammar.
Signed: Del Slane

```
grammar TableGen;

options {
        ASTLabelType=CommonTree;
        output=AST;
        k=2;
}

tokens {
        BLOCK;                                                          10
        CONDITIONAL;
        COORD;
        FUNC_CALL;
        FUNC_DEF;
        LIST_CONCAT;
        PROGRAM;
        TYPE_DECL;
        VAR_LOCATION;
        NUMERICAL_INDEX;
}                                                                       20

/* ====================== BEGIN LEXER ========================== */

/****************** Primitives **********************/
ASSIGNMENT_OP : '=' ;

ADDITIVE_OP : '+' | '-' ;

BOOL_OP : '<' | '>' | '<=' | '>=' | '==' | '!=';
                                                                        30
COLON : ':' ;

COMMA : ',' ;

COORD_SEPARATOR : '|' ;

DECIMAL_POINT : '.' ;

//DIGIT : '\u0030'..'\u0039' ;
fragment                                                                40
DIGIT : '0'..'9' ;
```

```
ELSE_KEYWORD : 'else' ;

ELSEIF_KEYWORD : 'elseif' ;

FOREACH_KEYWORD : 'foreach' ;

FOR_KEYWORD : 'for' ;

FUNC_KEYWORD : 'func' ;

HASH_ASSIGN : '>>' ;

IF_KEYWORD : 'if' ;

IN_KEYWORD : 'in' ;

L_BRACE : '{' ;

L_BRACKET : '[' ;

L_PAREN : '(' ;

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

LINE_TERMINATOR
        : ('\n' '\r') => '\n' '\r'
        | '\n'
        | '\r'
        ;

LOGICAL_OP
        : '&&'
        | '||'
        ;

ML_COMMENT_BEGIN : '/*';

ML_COMMENT_END : '*/';

MULTIPLICATIVE_OP : '*' | '/' ;

PUT_OP  : '->' ;


QUOTES : '"' ;

R_BRACE : '}' ;

R_BRACKET : ']' ;

R_PAREN : ')' ;
```

```
RETURN_KEYWORD : 'return' ;

SEMICOLON : ';' ;

SL_COMMENT : '//' ;                                                                    100

fragment
UNDERSCORE : '_' ;

VARTYPE : 'str' | 'num' | 'list';

// as at: http://www.columbia.edu/kermit/utf8-t1.html, verified OK from
//     http://www.unicode.org/charts/PDF/U0000.pdf
fragment
VALID_CHARS : '\u0020'..'\u007e' ;                                                     110

WHILE_KEYWORD : 'while' ;

WS : (' '|'\t') {$channel=HIDDEN; skip();} ;

Ml_Comment : ML_COMMENT_BEGIN (options {greedy=false;} : .*) ML_COMMENT_END {$channel=HIDDEN; skip();} ;
Sl_Comment : SL_COMMENT (options {greedy=false;} : .*) LINE_TERMINATOR {$channel=HIDDEN; skip();} ;

            // backslash
Escaped_char : '\u005c' ('"'| '-'| '+'| '/'| ',') ;                                    120

Identifier : UNDERSCORE? LETTER (LETTER | DIGIT | UNDERSCORE)* ;

String_literal : QUOTES (Escaped_char | ~('"' | '\u005c'))* QUOTES ;

Number_literal
        : '-'? DIGIT+ (DECIMAL_POINT DIGIT+)?;


/* ====================== END LEXER ============================== */               130


literal
        : Number_literal
        | String_literal
        ;

coordinate : L_BRACKET expression COORD_SEPARATOR expression R_BRACKET
                -> ^(COORD expression expression) ;
                                                                                       140
/****************** EVERY POSSIBLE expression *********************/

/* chain of expression statements to assign precedence */
expression : bool_expr1 ;

bool_expr1
        : (bool_expr2 -> bool_expr2)
                (LOGICAL_OP bool_expr2 -> ^(LOGICAL_OP $bool_expr1 bool_expr2) )*
        ;
```

```
                                                                                                                    150
bool_expr2
        : (add_expr -> add_expr)
                (BOOL_OP add_expr -> ^(BOOL_OP $boo$\ell$_expr2 add_expr))*
        ;

add_expr
        : (mult_expr -> mult_expr)
                ( ADDITIVE_OP mult_expr -> ^(ADDITIVE_OP $add_expr mult_expr) )*
        ;
                                                                                                                    160
mult_expr
        : (list_concat_expr -> list_concat_expr)
                ( MULTIPLICATIVE_OP list_concat_expr
                        -> ^(MULTIPLICATIVE_OP $mu$\ell$t_expr list_concat_expr) )*
        ;

list_concat_expr
        : (primary_expr -> primary_expr)
                ( COMMA primary_expr -> ^(COMMA $$\ell$ist_concat_expr primary_expr) )*
        ;                                                                                                           170

primary_expr
        : (par_expression -> par_expression)
        | (hash_key_specification) => (hash_key_specification -> hash_key_specification)
        | (identifier_reference -> identifier_reference)
        | (literal -> literal)
        ;

par_expression : L_PAREN expression* R_PAREN -> expression ;
                                                                                                                    180
identifier_reference
        : (Identifier L_PAREN) =>
                Identifier L_PAREN (func_call_param /*COMMA?*/)* R_PAREN -> ^(FUNC_CALL Identifier func_call_param*)
        | (Identifier DECIMAL_POINT) =>
                ( i1=Identifier DECIMAL_POINT i2=Identifier -> ^(DECIMAL_POINT $i1 $i2) )
        | (Identifier L_BRACKET) => ( Identifier L_BRACKET expression R_BRACKET
                        -> ^(NUMERICAL_INDEX Identifier expression) )
        | Identifier
        ;
                                                                                                                    190
hash_key_specification : Identifier HASH_ASSIGN hash_value
                -> ^(HASH_ASSIGN Identifier hash_value) ;

hash_value
        : identifier_reference -> identifier_reference
        | literal -> literal
        | par_expression -> par_expression
        ;

func_call_param                                                                                                     200
        : named_func_param
        | expression
        ;
```

```
named_func_param : Identifier COLON expression -> ^(COLON Identifier expression) ;

// either the location for the variable, or a table position–
//    essentially where one can put something or get something from
location
        : identifier_reference                                                    210
        | coordinate
        ;


/******************** STATEMENT TYPES ************************/

assignment : (VARTYPE COLON)? Identifier ASSIGNMENT_OP expression
                -> ^(ASSIGNMENT_OP ^(TYPE_DECL VARTYPE?) Identifier expression);

additional_elements: COMMA expression -> expression ;                             220

t_if : IF_KEYWORD expression block -> ^(IF_KEYWORD expression block) ;
t_elseif : ELSEIF_KEYWORD expression block -> ^(ELSEIF_KEYWORD expression block) ;
t_else : ELSE_KEYWORD block -> ^(ELSE_KEYWORD block) ;

function_definition : FUNC_KEYWORD Identifier func_param* block
        -> ^(FUNC_DEF Identifier func_param* block ) ;

func_param
        : typed_func_param                                                        230
        | untyped_func_param
        ;

typed_func_param : VARTYPE COLON Identifier -> ^(COLON VARTYPE Identifier) ;
untyped_func_param : Identifier -> Identifier ;

for_loop : FOR_KEYWORD a1=assign_or_expr SEMICOLON expression SEMICOLON a2=assign_or_expr block
                -> ^(FOR_KEYWORD $a1 expression $a2 block) ;

assign_or_expr : ( ((assignment)=>assignment -> assignment) | (expression -> expression)) ;   240

// for each $i1 in $i2 , do BLOCK
// allowing the target of the IN keyword to be an arbitrary expression – check at runtime
foreach_loop :
        FOREACH_KEYWORD (VARTYPE COLON)? Identifier IN_KEYWORD expression block
                        -> ^(FOREACH_KEYWORD ^(TYPE_DECL VARTYPE?) Identifier expression block )
        ;

while_loop : WHILE_KEYWORD expression block -> ^(WHILE_KEYWORD expression block) ;
                                                                                  250
return_stmt : RETURN_KEYWORD expression -> ^(RETURN_KEYWORD expression) ;

putop : expression PUT_OP location -> ^(PUT_OP expression location) ;

/******************** HIGH-LEVEL **********************/
program : LINE_TERMINATOR* statement* -> ^(PROGRAM statement*);
```

```
statement
        :
                /* syntactic predicate on putop scans at most one line because the '->'
                 * denotes a putop rather than something else
                 */
                ( ((putop) => putop -> putop)
                | (assignment -> assignment)
                | (t_if -> t_if)
                | (t_elseif -> t_elseif)
                | (t_else -> t_else)
                | (function_definition -> function_definition)
                | (for_loop -> for_loop)
                | (foreach_loop -> foreach_loop)
                | (while_loop -> while_loop)
                | (return_stmt -> return_stmt)
                | (expression -> expression)
                | (block -> block)
                ) ( LINE_TERMINATOR+ -> $statement )
        ;

block : L_BRACE LINE_TERMINATOR+ statement* R_BRACE -> ^(BLOCK statement*) ;
```

8.1.3 main/TableGen.java

This is our main java executable.
Signed: Del Slane

```
package main;

import java.io.*;
import java.util.*;
import org.antlr.runtime.*;
import org.antlr.runtime.tree.*;

import excep.*;

import gram.TableGenLexer;
import gram.TableGenParser;
import symtb.*;

public class TableGen {

        public static boolean debug = false;

        public static void main(String[] args) {

                try {
                        CharStream cs = new ANTLRFileStream(args[0]);
                        TableGenLexer lexer = new TableGenLexer(cs);
```

```
                CommonTokenStream tokens = new CommonTokenStream();
                tokens.setTokenSource(lexer);

                TableGenParser parser = new TableGenParser(tokens);

                TableGenParser.program_return r = parser.program();
                CommonTree t = (CommonTree)r.getTree();                                    30

                if (debug)
                        System.out.println(t.toStringTree());

            TGAbstractSymbolTable abstractSymbolTable = new TGAbstractSymbolTable (t);
            abstractSymbolTable.build();
            if (debug)
                    abstractSymbolTable.printScopes();

            if (debug)                                                                     40
                    System.out.println("\n-----------------------------------------------\n");

            Interpreter interpreter = new Interpreter(t, abstractSymbolTable);
            interpreter.interpret();
            }
            catch (IOException e) {
                    System.out.println("Error getting data from file: " + e.getMessage());
                    System.exit(1);
            }
            catch (RecognitionException e) {                                           50
                    System.out.println("Recognition Exception received");
                    System.exit(2);
            }

    }

}
```

8.1.4 main/Interpreter.java

This is our main Table Generator file.
Signed: Del Slane

```
package main;

import excep.*;
import gram.*;
import symtb.*;
import java.util.*;
import org.antlr.runtime.tree.*;

public class Interpreter {
```

```
public static boolean debug = TableGen.debug;
/* values for referring to the type of a symbol – need to have
 * a distinct hash type too because not everything is a hash when
 * talking about values
 */
public static final int INTEGER_SYMBOL_TYPE = 1;
public static final int DOUBLE_SYMBOL_TYPE = 2;
public static final int STRING_SYMBOL_TYPE = 3;
public static final int HASH_SYMBOL_TYPE = 4;

private TGAbstractSymbolTable symbols;
private CommonTree ast;
private LinkedList<TGScope> symbolTable;
private Vector<Vector<String>> outputTable;

private Stack<TGHash> returnValues;

// These are needed so that we don't print a bunch of
// blank at the end.
private int maxRow;
private int maxCol;

/* needed to check to see if the previous statement executed was an if/elseif statement
 * so an 'else' block can be ignored as necessary
 */
private boolean ifJustExecuted = false;

/*
 * A global boolean variable that allows an executing loop or scope to simply
 * stop executing when a function returns
 */
private boolean functionIsReturning = false;

private int initialTableSize = 10;
private int tableExpansionSize = 100;

public Interpreter(CommonTree ast, TGAbstractSymbolTable symbols)
{
        this.ast = ast;
        this.symbols = symbols;
        this.symbolTable = new LinkedList<TGScope>();
        this.returnValues = new Stack<TGHash>();

        this.outputTable = new Vector<Vector<String>> (initialTableSize, tableExpansionSize);

        for (int i = 0; i < outputTable.capacity(); i++){
            this.outputTable.add(new Vector<String> (initialTableSize, tableExpansionSize));
            for (int j = 0; j < outputTable.get(i).capacity(); j++)
                outputTable.get(i).add("");
        }

        this.maxRow = -1;
        this.maxCol = -1;
}
```

```
public void interpret() {
        this.pushScope(this.symbols.getGlobalScope());
        this.interpretBlock(ast);
}

/**                                                                              70
 * This method interprets the AST subtree it is given
 * @param tree the AST subtree to interpret
 * @param treeIsBlock pass 'true' if only the children of the passed tree are to be
 *                    executed as in a block statement or 'false' if the tree is just a
 *                    statement itself
 */
private void interpretBlock(Tree tree) {
        if (debug)
                System.out.println("dbg: Int.interpretBlock-- #children="+tree.getChildCount());
        for (int childNumber = 0; childNumber < tree.getChildCount(); childNumber++) {   80
                Tree currentChild = tree.getChild(childNumber);
                if (debug)
                        System.out.println("dbg: Int.interpretBlock-- child="+currentChild.toStringTree());
                interpretStatement(currentChild);
                if (functionIsReturning)
                    return;
        }

        /* if the current scope has been entirely executed and it is the global scope */
        if (this.symbolTable.size() == 1 && this.symbolTable.get(0).equals(symbols.getGlobalScope()))   90
                outputTable();
}

private void interpretStatement(Tree statement) {
        if (statement.getType() == TableGenParser.ASSIGNMENT_OP)
                doAssignmentOp(statement);
        else if (statement.getType() == TableGenParser.PUT_OP)
                doPutOp(statement);
        /*
         * this case covered by the final statement handling expressions . . . I think.   100
         * else if (statement.getType() == TableGenParser.FUNC_CALL)
                doFuncCall(statement, false);
        */
        else if (statement.getType() == TableGenParser.WHILE_KEYWORD)
                doWhileLoop(statement);
        else if (statement.getType() == TableGenParser.FOR_KEYWORD)
                doForLoop(statement);
        else if (statement.getType() == TableGenParser.FOREACH_KEYWORD)
                doForeachLoop(statement);
        else if (statement.getType() == TableGenParser.IF_KEYWORD)                      110
                doIf(statement);
        else if (statement.getType() == TableGenParser.ELSEIF_KEYWORD)
                //internal checking of ifJustExecuted flag
                doElseif(statement);
        else if (statement.getType() == TableGenParser.ELSE_KEYWORD)
                //internal checking of ifJustExecuted flag
                doElse(statement);
```

```
else if (statement.getType() == TableGenParser.RETURN_KEYWORD) {
        if (debug)
                System.out.println ("Return statement found. . .about to evaluate this: " +
                                    statement.getChild(0).toStringTree ());
    TGHash returnValue = evaluateExpression(statement.getChild(0));
    functionIsReturning = true;
            if (debug)
                System.out.println ("RValue " + returnValue.toString ());
    this.returnValues.push(returnValue.clone());
}

        /* handle expressions */
        else if (
                                statement.getType() == TableGenParser.ADDITIVE_OP ||
                                statement.getType() == TableGenParser.MULTIPLICATIVE_OP ||
                                statement.getType() == TableGenParser.COMMA ||
                                statement.getType() == TableGenParser.FUNC_CALL ||
                                statement.getType() == TableGenParser.BOOL_OP
                        )
        {
                evaluateExpression(statement);
        }

        if (statement.getType() != TableGenParser.IF_KEYWORD &&
            statement.getType() != TableGenParser.ELSEIF_KEYWORD)
                this.ifJustExecuted = false;
}

/* ~!~Del */
private void doAssignmentOp(Tree assignmentTree)
{
        String lValueName = assignmentTree.getChild(1).getText();

        try {
                TGHash symbol = getSymbol(lValueName, assignmentTree.getChild(1).getLine());
                TGHash value = evaluateExpression(assignmentTree.getChild(2));
                symbol.setValue(value);

                if (debug)
                        System.out.println("dbg: doAssignmentOp-- value type as TGNum="
                                           + symbol.getFirstSymbol().getRuntimeType());

                if (debug)
                        System.out.println("dbg: doAssignmentOp-- assinging value '" + value.toString() +
                                           "' to variable '" + lValueName +"'");
        }
        catch (TGSymbolNotFoundException e) {
                System.out.println(e.getMessage());
                System.exit(-1);
        }
        catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(), assignmentTree.getLine()));
                System.exit(-1);
        }
```

```
}

/*
 * Currently only handles coordinates, not identifiers as lists
 *
 * ~!~Del
 */
private void doPutOp(Tree putOpTree) {
        TGHash value = evaluateExpression(putOpTree.getChild(0));                                    180
        TGHash row = evaluateExpression(putOpTree.getChild(1).getChild(0));
        TGHash col = evaluateExpression(putOpTree.getChild(1).getChild(1));

        try {
                if (debug)
                        System.out.println("dbg doPutOp-- putting "+value.toString()+" into " +
                                        "["+row.getFirstSymbol().getIntegerValue()+"|"+
                                        col.getFirstSymbol().getIntegerValue()+"]");

                putInOutputTable(value,                                                              190
                                row.getFirstSymbol().getIntegerValue(),
                                col.getFirstSymbol().getIntegerValue());
        }
        catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(), putOpTree.getLine()));
        }
        catch (NoSuchElementException e) {
                System.out.println("Invalid coordinates '["+row.toString()+"|"+col.toString()+"]' " +
                                "for coordinates on line " + putOpTree.getLine());
                System.exit(-1);                                                                     200
        }
}

private void putInOutputTable(TGHash value, int row, int col) {
    if (row > maxRow){
        if (row > outputTable.size()){
            for (int i = outputTable.size(); i <= row; i++){
                outputTable.add(new Vector<String> (initialTableSize, tableExpansionSize));
                for (int j = 0; j < outputTable.get(i).capacity(); j++)
                    outputTable.get(i).add("");                                                      210
            }
        }
        maxRow = row;
    }

    if (col > maxCol){
        if (col > outputTable.get(row).size()){
            for (int i = 0; i < outputTable.size(); i++){
                for (int j = outputTable.get(row).size(); j <= col; j++){
                    outputTable.get(i).add("");                                                      220
                }
            }
        }
        maxCol = col;
    }
```

```
    this.outputTable.get(row).setElementAt(value.toString(), col);
}

/**
 * Executes the function call specified and pushes a return value onto the local variable
 * returnStack if required, throwing an error on failure
 * @param funcCallTree the tree with the FUNC_CALL node as the root
 * @param returnRequired a boolean value indicating if a return values should be
 *                       pushed onto the stack
 * @return a boolean value indicating if a return value was pushed onto the return stack
 *
 * ~!~Del
 */
private boolean doFuncCall(Tree funcCallTree)
                throws TGMissingReturnValueException
{
        String funcName = funcCallTree.getChild(0).getText();

        /* intercept reserved functions */
        if (funcName.charAt(0) == '_') {
                // _tail function always returns something
                if (funcName.equals ("_tail")) {
            doTailFunc (funcCallTree);
                        return true;
                }

                // _length functions always returns something.
                else if (funcName.equals ("_length")) {
                        doLengthFunc (funcCallTree);
                        return true;
                }

                else if (funcName.equals ("_avg")) {
                        doAvgFunc (funcCallTree);
                        return true;
                }

                else if (funcName.equals ("_sum")) {
                        doSumFunc (funcCallTree);
                        return true;
                }

                else if (funcName.equals ("_round")) {
                        doRoundFunc (funcCallTree);
                        return true;

                }
                throw new RuntimeException("Function name beginning with '_' not a valid " +
                                "reserved function\n");
        }



        try {
```

```
    //String funcName = funcCallTree.getChild(0).getText();                    280

    Tree funcDefTree = this.getFunctionTree(funcName);
    Tree blockTree = funcDefTree.getChild(funcDefTree.getChildCount() - 1);

    TGScope functionScope = this.symbols.getFunctionScope(funcName);

    if (functionScope == null)
        throw new RuntimeException("Compiler Error: could not get scope of function '" +
                funcName + "'");
                                                                             290
    /* initialize parameters */

    /* case for first being all named parameters */
    if (funcCallTree.getChildCount() > 1 &&
                funcCallTree.getChild(1).getType() == TableGenParser.COLON)
    {
        for (int i = 1; i < funcCallTree.getChildCount(); i++) {
            /* easiest to hold value in TGHash because it can associate
             * names and values already */
            String paramName = funcCallTree.getChild(i).getChild(0).getText();   300

            try {
                TGHash value = evaluateExpression(funcCallTree.getChild(i).getChild(1));
                setValueInScope(paramName, value, HASH_SYMBOL_TYPE, functionScope);
            }
            catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(),
                                funcCallTree.getLine()).toString());
                System.exit(-1);
            }                                                                310
            if (debug)
                System.out.println("dbg: Int.doFuncCall, all named params, completed i="+
        }
    }

    /* case for first all being unnamed parameters, symbols matched up as OK in
     * static-semantic analysis
*/
    else {
        int funcTreeParams = funcCallTree.getChildCount() - 1;                320
        TGHash[] symbols = functionScope.getSymbols();

        for (int i = 0; i < funcTreeParams; i++) {
            // i+1 because we must exclude the function name
            try {
                TGHash paramValue = evaluateExpression(funcCallTree.getChild(i+1));
                symbols[i].setValue(paramValue);
            }
            catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(),          330
                                funcCallTree.getLine()).toString());
                System.exit(-1);
            }
```

```
                }
            }

            /* execute the function, return value if necessary –
             * interpret ignores return statements
             */
            this.pushScope(functionScope);                                          340
            interpretBlock(blockTree);
            if (functionIsReturning) {
                // Functions with nothing in them in need to
                // return something anyway.
                if (returnValues.isEmpty ())
                    returnValues.push (new TGHash ());
                functionIsReturning = false;
            }

            this.popScope();                                                        350

            //if (blockTree.toStringTree ().contains ("FUNC_CALL"))
            //      return true;

            return true;
        }
        catch (TGSymbolNotFoundException e) {
            System.out.print(e.getMessage());
            System.out.println(" from doFuncCall.");
            System.exit(-1);                                                        360
        }
        //if (debug)
            //System.out.println("dbg: doFuncCall– top return stack value="+this.returnValues.get(0));
        return false;
    }

    private void doTailFunc (Tree funcCallTree) {
        // Tail only takes a list as input.
        if (funcCallTree.getChildCount() != 2) {
            System.out.println("Error: Call to _tail can only take one argument.");  370
            System.exit (-1);
        }

        TGHash tail = evaluateExpression(funcCallTree.getChild(1));
        this.returnValues.push(tail.getTail());
    }

    private void doLengthFunc (Tree funcCallTree) {
        if (funcCallTree.getChildCount() != 2) {
            System.out.println("Error: Call to _length can only take one argument; I think you gave it "  1
                               + funcCallTree.getChildCount());
            System.exit (-1);
        }

        TGHash len = evaluateExpression(funcCallTree.getChild(1));
        this.returnValues.push(len.getLength ());
    }
```

```
private void doAvgFunc(Tree funcCallTree){
if (funcCallTree.getChildCount() != 2) {                                                    390
        System.out.println("Error: Call to _avg can only take one argument; I think you gave it "
                        + funcCallTree.getChildCount());
        System.exit (-1);
}
                TGHash arg = evaluateExpression(funcCallTree.getChild(1));
                TGHash value = new TGHash();
                double sum = 0;
                boolean is_double = false;
                try {
                        for (int i = 0; i < arg.getSymbolCount(); i++){                     400
                                if (arg.getSymbol(i).isDouble() || arg.getSymbol(i).isRuntimeDouble()){
                                        is_double = true;
                                        sum += arg.getSymbol(i).getDoubleValue().doubleValue();
                                }
                                else
                                        sum += arg.getSymbol(i).getIntegerValue().intValue();
                        }
                        if (!is_double && (((int)sum) % arg.getSymbolCount() != 0))
                                is_double = true;
                                                                                            410
                        if (is_double)
                                value.addAsFirst(new TGSymbol(new TGNum(sum / arg.getSymbolCount())));
                        else
                                value.addAsFirst(new TGSymbol(new TGNum((int)(sum / arg.getSymbolCount()))));

                        this.returnValues.push(value);
                }
                catch (TGTypeException e){

                }                                                                           420
}

private void doSumFunc(Tree funcCallTree){
        if (funcCallTree.getChildCount() != 2) {
                System.out.println("Error: Call _sum can only take one argument; I think you gave it "
                                + funcCallTree.getChildCount());
                System.exit (-1);
        }
        TGHash arg = evaluateExpression(funcCallTree.getChild(1));
        TGHash value = new TGHash();                                                        430
        double sum = 0;
        boolean is_double = false;
        try {
                for (int i = 0; i < arg.getSymbolCount(); i++){
                        if (arg.getSymbol(i).isDouble() || arg.getSymbol(i).isRuntimeDouble()){
                                is_double = true;
                                sum += arg.getSymbol(i).getDoubleValue().doubleValue();
                        }
                        else
                                sum += arg.getSymbol(i).getIntegerValue().intValue();       440
                }
```

```
                if (is_double)
                        value.addAsFirst(new TGSymbol(new TGNum(sum)));
                else
                        value.addAsFirst(new TGSymbol(new TGNum((int)sum)));

                this.returnValues.push(value);
        }
        catch (TGTypeException e){                                              450

        }
}

private void doRoundFunc (Tree funcCallTree) {
        if (funcCallTree.getChildCount() < 2 || funcCallTree.getChildCount() > 3) {
                System.out.println("Error: Round function can only take one or two agruments; I think you gave
                                        + funcCallTree.getChildCount());
                System.exit (-1);
        }                                                                       460

        try {
                TGHash roundTo = new TGHash ();
                if (funcCallTree.getChildCount() == 3) {
                        roundTo = evaluateExpression (funcCallTree.getChild(2));
                } else {
                        roundTo.addAsFirst(new TGSymbol("temp", new TGNum(100.0))); // Default to 100.
                }
                TGHash value = new TGHash ();
                value = evaluateExpression (funcCallTree.getChild(1));          470

                        /* value is an integer; don't round */
                if (value.getFirstSymbol().isRuntimeInt())
                        this.returnValues.push(value);

                /* at least one operand is a double – the result is also a double */
                else {
                        double val = value.getFirstSymbol().getDoubleValue();
                        double rnd = roundTo.getFirstSymbol().getDoubleValue();
                                                                                480
                        double ret = Math.round (val * rnd) / rnd;

                        TGHash rounded = new TGHash ();
                        rounded.addAsFirst(new TGSymbol("temp", new TGNum(ret)));
                        this.returnValues.push(rounded);
                }
        } catch (TGTypeException e) {
                System.out.println("Error: types to not match in doRoundFunc.");
        }
                                                                                490

}

private void doWhileLoop(Tree whileTree){
TGHash result;
```

```
Tree test = whileTree.getChild(0);
Tree block = whileTree.getChild(1);
TGScope whileScope = this.symbolTable.peek().getNextChildScope();
try {
    while (true){                                                                    500
        result = evaluateExpression(test);
        int i = result.getFirstSymbol().getNumValue().getIntValue();
        if (i == 1) {
            whileScope.resetScopeIterator();
            this.pushScope(whileScope);
            interpretBlock(block);
            this.popScope();
            if (functionIsReturning)
                return;
        }                                                                            510
        else
            break;
    }
}
catch (TGTypeException ex){
    System.out.println(new TGTypeException(ex.getMessage(),test.getLine()).toString());
    System.exit(-1);
}
}
                                                                                     520
private void doForLoop(Tree forTree){
        if (debug)
                System.out.println("dbg: doForLoop-- top scope="+this.symbolTable.peek());
        TGScope forScope = this.symbolTable.peek().getNextChildScope();
        if (debug)
                System.out.println("dbg: doForLoop-- forScope="+forScope.toString());

        pushScope(forScope);
        TGHash result;
        Tree init = forTree.getChild(0);                                             530
        Tree test = forTree.getChild(1);
        Tree count = forTree.getChild(2);
        Tree block = forTree.getChild(3);

try {
        interpretStatement(init);
        while (true){
                result = evaluateExpression(test);
                int i = result.getFirstSymbol().getNumValue().getIntValue();
        if (i == 1) {                                                                540
                /* reset the child scope iterator for each loop execution */
                this.symbolTable.peek().resetScopeIterator();
            interpretBlock(block);
            interpretStatement(count);
            if (functionIsReturning)
                break;
        }
        else
            break;
```

```
    }
}
catch (TGTypeException ex){
     System.out.println(new TGTypeException(ex.getMessage(), test.getLine()).toString());
     System.exit(-1);
}
popScope();
}

private void doForeachLoop(Tree foreachTree) {
        try {
                TGScope foreachScope = this.symbolTable.peek().getNextChildScope();
                TGHash itemList = getSymbol(foreachTree.getChild(2).getText(), foreachTree.getLine());
                this.pushScope(foreachScope);
                TGHash iterator = getSymbol(foreachTree.getChild(1).getText(), foreachTree.getLine());

                if (debug)
                        System.out.println("dbg: doForeachLoop-- itemList="+itemList.toString()+
                                        ", iterator="+iterator.toString());

                TGSymbol[] listSymbols = itemList.getSymbols();
                for (int i = 0; i < listSymbols.length; i++) {
                        TGHash itrVal = new TGHash();
                        itrVal.addAsLast(listSymbols[i]);
                        iterator.setValue(itrVal);
                        foreachScope.resetScopeIterator();

                        interpretBlock(foreachTree.getChild(3));
                        if (functionIsReturning)
                                break;
                }

                this.popScope();
        }
        catch (TGSymbolNotFoundException e) {
                System.out.println(e.getMessage());
                System.exit(-1);
        }
        catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(), foreachTree.getLine()).toString());
                System.exit(-1);
        }

}

private void doIf (Tree ifTree) {
    TGHash result;
    Tree test = ifTree.getChild(0);
    Tree block = ifTree.getChild(1);
    try {
        result = evaluateExpression(test);
        int i = result.getFirstSymbol().getNumValue().getIntValue();
        if (i == 1) {
                pushScope(symbolTable.peek().getNextChildScope());
```

```
                interpretBlock(block);
                popScope();
                this.ifJustExecuted = true;

                if (functionIsReturning)
                        return;
        }
        else
                symbolTable.peek().getNextChildScope(); // discard the if's scope
    } catch (TGTypeException ex){
        System.out.println(ex.getMessage());
        System.exit(-1);
    }
}

private void doElseif(Tree elseifTree) {
        if (!this.ifJustExecuted)
                doIf(elseifTree);
}

private void doElse(Tree elseTree) {
        this.pushScope(this.symbolTable.peek().getNextChildScope());
        if (!this.ifJustExecuted)
                interpretBlock(elseTree.getChild(0));
        this.popScope();
}

/**
 * Gets the FUNC_DEF node corresponding to the function with the specified name
 * @param functionName the name of the function to find in the tree
 * @return a tree with the root being the FUNC_DEF node of the function declaration
 *                  for the function of the specified name
 * ~!~Del
 */
private Tree getFunctionTree(String functionName)
                throws TGSymbolNotFoundException
{
        int i;
        for (i = 0; i < this.ast.getChildCount(); i++) {
                Tree funcDefCandidate = this.ast.getChild(i);
                if (funcDefCandidate.getType() == TableGenParser.FUNC_DEF &&
                                funcDefCandidate.getChild(0).getText().equals(functionName))
                {
                        return funcDefCandidate;
                }
        }

        throw new TGSymbolNotFoundException(functionName);
}

/**
 * Only looks inside the current scope for setting the value of the symbol with the
 * specified name – created primarily for initializing parameters before a function call.
 * This function assumes that the scope to search has already been pushed onto the
```

```
 * symbol table and is on the top of the symbol table stack.
 * @param symbolName the name of the symbol to set the value of
 * @param value the value to which to set the symbol
 * @param type the type of the passed Object value – as in TGSymbol
 * @throws TGSymbolNotFoundException when the symbol is not found in the current scope
 * @throws TGTypeException when setting the value fails because of invalid types
 *
 * ~!~Del
 */
private void setValueInScope(String symbolName, Object value, int type, TGScope scope)
        throws TGSymbolNotFoundException, TGTypeException
{
    TGHash[] symbols = scope.getSymbols();

    for (int i = 0; i < symbols.length; i++) {
        if (symbols[i].getName().equals(symbolName)) {
            if (type == INTEGER_SYMBOL_TYPE)
                symbols[i].setValue((Integer)value);
            else if (type == DOUBLE_SYMBOL_TYPE)
                symbols[i].setValue((Double)value);
            else if (type == STRING_SYMBOL_TYPE)
                symbols[i].setValue((String)value);
            else if (type == HASH_SYMBOL_TYPE)
                symbols[i].setValue((TGHash)value);
            else
                throw new RuntimeException("Invalid type passed " +
                                           "to Interpreter.setCurrentScopeSymbolValue");
        }
    }
}

/**
 * Use this method to set the value of a symbol. It will throw the appropriate
 * exceptions based on the symbol table values and the type of the parameter
 * @param symbolName the name of the symbol of which to change the value
 * @param value the value to which to set the symbol– this may be of type Integer,
 *              Double, String or TGHash
 * @param type the static final flag as declared at the top of the file corresponding to
 *              the type of the value passed in
 * @throws TGSymbolNotFoundException when the symbol could not be found in the symbol table
 * @throws TGTypeException when the underlying assignment implementation detects a type issue
 *              due to programmer-imposed type constraints in the program
 *
 * ~!~Del
 */
private void setSymbolValue(String symbolName, Object value, int type)
        throws TGSymbolNotFoundException, TGTypeException
{
    boolean found = false;
    TGScope[] scopes = new TGScope[this.symbolTable.size()];
    scopes = this.symbolTable.toArray(scopes);

    for (int i = 0; i < scopes.length; i++) {
        TGHash[] symbols = scopes[i].getSymbols();
```

```
                for (int j = 0; j < symbols.length; j++) {
                        if (symbols[j].getName().equals(symbolName)) {
                                found = true;
                                if (type == INTEGER_SYMBOL_TYPE)
                                        this.symbolTable.get(i).getSymbol(j).setValue((Integer)value);
                                else if (type == DOUBLE_SYMBOL_TYPE)
                                        this.symbolTable.get(i).getSymbol(j).setValue((Double)value);
                                else if (type == STRING_SYMBOL_TYPE)                                    720
                                        this.symbolTable.get(i).getSymbol(j).setValue((String)value);
                                else if (type == HASH_SYMBOL_TYPE)
                                        this.symbolTable.get(i).getSymbol(j).setValue((TGHash)value);
                                else
                                        throw new RuntimeException("Invalid type passed to " +
                                                        "Interpreter.setSymbolValue");
                        }
                }

                if (found)                                                                              730
                        break;
        }

        if (!found)
                throw new TGSymbolNotFoundException(symbolName);
}

/**
 * Gets the TGSymbol of the corresponding name from the symbol table
 * @param symbolName the name of the symbol to return                                              740
 * @param lineNumber the line number of the program on which the symbol is referenced
 * @return the TGSymbol object with the specified name
 * @throws TGSymbolNotFoundException if the symbol is not currently
 *                      visible in the symbol table
 *
 * ~!~Del
 */
private TGHash getSymbol(String symbolName, int lineNumber)
                throws TGSymbolNotFoundException
{                                                                                                      750
        TGHash result = null;
        TGScope[] scopes = new TGScope[this.symbolTable.size()];
        scopes = this.symbolTable.toArray(scopes);

        for (int i = 0; i < scopes.length; i++) {
                TGHash[] symbols = scopes[i].getSymbols();

                for (int j = 0; j < symbols.length; j++) {
                        if (symbols[j].getName().equals(symbolName))
                                result = this.symbolTable.get(i).getSymbol(j);                         760
                }

                if (result != null)
                        break;
        }
```

```
        if (result == null)
                throw new TGSymbolNotFoundException(symbolName, lineNumber);
        return result;
}

private void pushScope(TGScope scope) {
        this.symbolTable.addFirst(scope);
}

/**
 * Implementation of the "." operator to get a value out of a hash
 * @param hashName the name of the actual symbol to look inside
 * @param hashKey the hash key of the value inside the symbol's element list
 * @param lineNumber the line number the request is coming from for proper
 *                   formation of error messages
 * @return a new TGHash object holding the value extracted from the specified hash
 *
 * ~!~Del
 */
private TGHash lookupSymbolInHash(String hashName, String hashKey, int lineNumber)
                throws TGSymbolNotFoundException
{
        TGHash result = null;
        TGSymbol element;

        try {
                result = getSymbol(hashName, lineNumber);
                element = result.getElement(hashKey);
                if (element.isRuntimeDouble())
                        result.setValue(element.getDoubleValue());
                else if (element.isRuntimeInt())
                        result.setValue(element.getIntegerValue());
                else if (element.isRuntimeString())
                        result.setValue(element.getStringValue());
                else {
                        System.out.println("Unrecognized runtime type in lookupSymboInHash on line "
                                        + lineNumber);
                        System.exit(-1);
                }
        }
        catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(), lineNumber).toString());
                System.exit(-1);
        }
        catch (TGSymbolNotFoundException e) {
                System.out.print(e.getMessage());
                System.out.println (" in lookupSymbolInHash.");
                System.exit (-1);
        }

        return result;
}
```

```
/**                                                                    820
 * Discards the scope on the top of the symbolTable scope stack
 */
private void popScope() {
        /* the scope is gone, so zero all the values here so they are ready
         * for the next time the scope is used
         */
        TGScope popped = this.symbolTable.removeFirst();
        TGHash[] symbols = popped.getSymbols();

        for (int i = 0; i < symbols.length; i++)                       830
                symbols[i].zero();
}

/**
 * Evaluates the given expression tree
 * @param expressionTree the tree representing the expression to evaluate
 * @return a temporary TGHash variable
 */
private TGHash evaluateExpression(Tree expressionTree)
                throws TGMissingReturnValueException                   840
{
        TGHash result = null, x = null, y = null;

        try {
            if (expressionTree.getType() == TableGenParser.FUNC_CALL) {
                boolean pushedReturnValue = doFuncCall(expressionTree);
                if (debug)
                        System.out.println ("Finished FuncCall.");
                if (pushedReturnValue)
                        // This only happens when function has a return  850
                        // statement. Functions without return statements
                        // don't push anything onto the stack, hense why
                        // we have to do the following.
                        if (!this.returnValues.isEmpty ())
                                result = this.returnValues.pop();
                        if (debug)
                                System.out.println ("Return value popped: " +
                                result.toString () + " " + returnValues.empty());
            }
                                                                       860
            else if (expressionTree.getType() == TableGenParser.Identifier) {
                result = getSymbol(expressionTree.getText(), expressionTree.getLine());
            }

            else if (expressionTree.getType() == TableGenParser.Number_literal){
                result = new TGHash();
                try {
                int value = Integer.parseInt(expressionTree.getText());
                result.addAsFirst(new TGSymbol("temp", new TGNum(value)));
        }                                                              870
        catch (NumberFormatException es) {
                try {
                double value = Double.parseDouble(expressionTree.getText());
```

```
        result.addAsFirst(new TGSymbol("temp", new TGNum(value)));
        }
        catch (NumberFormatException ex) {
                throw new RuntimeException("Unable to parse '" +
                  expressionTree.getText() +
                  "' as a number literal on line " + expressionTree.getLine());
   }                                                                              880
}
}

else if (expressionTree.getType() == TableGenParser.String_literal) {
        result = new TGHash("temp", TGSymbol.TG_STR_TYPE);
    String value = expressionTree.getText();
    result.addAsFirst(new TGSymbol("temp", value));
}

else if (expressionTree.getType() == TableGenParser.ADDITIVE_OP) {              890
        result = new TGHash("temp", TGSymbol.TG_NUM_TYPE);
    x = evaluateExpression(expressionTree.getChild(0));
    y = evaluateExpression(expressionTree.getChild(1));
    if (expressionTree.getText().equals("+"))
        result = add(x, y);
    else
        result = subtract(x, y);

}
                                                                                900
else if (expressionTree.getType() == TableGenParser.MULTIPLICATIVE_OP) {
        result = new TGHash("temp", TGSymbol.TG_NUM_TYPE);
    x = evaluateExpression(expressionTree.getChild(0));
    y = evaluateExpression(expressionTree.getChild(1));
    if (expressionTree.getText().equals("*"))
        result = mult(x, y);
    else
        result = divide(x, y);
}
                                                                                910
else if (expressionTree.getType() == TableGenParser.BOOL_OP) {
        result = new TGHash("temp", TGSymbol.TG_NUM_TYPE);
    x = evaluateExpression(expressionTree.getChild(0));
    y = evaluateExpression(expressionTree.getChild(1));
    result = compare(x, y, expressionTree.getText());
}

else if (expressionTree.getType() == TableGenParser.LOGICAL_OP) {
        result = new TGHash("temp", TGSymbol.TG_NUM_TYPE);
    x = evaluateExpression(expressionTree.getChild(0));                         920
    y = evaluateExpression(expressionTree.getChild(1));

    if (!x.getFirstSymbol().isRuntimeNum() || !y.getFirstSymbol().isRuntimeNum())
        throw new TGTypeException("attempted operation includes a non-number value");
    int left = x.getFirstSymbol().getIntegerValue().intValue();
    int right = y.getFirstSymbol().getIntegerValue().intValue();
    if (expressionTree.getText().equals("&&")){
```

```
        if ((left == 1) && (right == 1))
            result.addAsFirst(new TGSymbol("temp", new TGNum(1)));
        else
            result.addAsFirst(new TGSymbol("temp", new TGNum(0)));
    }
    else if (expressionTree.getText().equals("||")){
        if ((left == 1) || (right == 1))
            result.addAsFirst(new TGSymbol("temp", new TGNum(1)));
        else
            result.addAsFirst(new TGSymbol("temp", new TGNum(0)));
    }
}

else if (expressionTree.getType() == TableGenParser.DECIMAL_POINT) {
        String hashName = expressionTree.getChild(0).getText();
        String hashKey = expressionTree.getChild(1).getText();
        if (debug)
                System.out.println("dbg: evaluateExpression-- inside DECIMAL_POINT case");
        result = lookupSymbolInHash(hashName, hashKey, expressionTree.getLine());
        if (debug)
                System.out.println("\t\t result="+result.toString());
}

else if (expressionTree.getType() == TableGenParser.NUMERICAL_INDEX) {
        result = new TGHash();
        String listName = expressionTree.getChild(0).getText();
        TGHash index = evaluateExpression(expressionTree.getChild(1));

        if (index.getSymbolCount() != 1)
                throw new TGTypeException("array index access with a value that does not have "+
                        "exactly 1 element");
        if (index.getFirstSymbol().getRuntimeType() != TGSymbol.TG_NUM_TYPE)
                throw new TGTypeException("array index access with an expression that does not " + 9
                        "evaluate to a numeric value");
        if (!index.getFirstSymbol().isRuntimeInt())
                throw new TGTypeException("array index access with an expression that evaluates " +
                        "do a fractional number rather than an integer");

        TGHash symbol = getSymbol(listName, expressionTree.getLine());
        int indexValue = index.getFirstSymbol().getIntegerValue();
        if (debug)
                System.out.println("dbg: evaluateExpression--NUMERICAL_INDEX, symbol="+ symbol.toString(
        if (indexValue >= symbol.getSymbolCount())
                throw new TGTypeException("array index ("+indexValue+") exceeds # of elements "+
                        "in the array ("+symbol.getSymbolCount()+")");

        TGSymbol value = symbol.getSymbol(indexValue);
        result.addAsLast(value);
}

else if (expressionTree.getType() == TableGenParser.HASH_ASSIGN) {
        if (debug)
                System.out.println("dbg: evaluateExpression-- inside HASH_ASSIGN case");
```

```
                result = new TGHash();
                String hashKey = expressionTree.getChild(0).getText();
                TGHash value = evaluateExpression(expressionTree.getChild(1));

                /* change this to allow for lists of lists */
                if (value.getSymbolCount() != 1) {
                        System.out.println("Invalid expression after list value for key '" +
                                        hashKey + "'");
                        System.exit(-1);                                                    990
                }

                value.getFirstSymbol().setKey(hashKey);
                result.addAsLast(value.getFirstSymbol());

                if (debug)
                        System.out.println("dbg: evaluateExpression-- ending HASH_ASSIGN case");
        }

        else if (expressionTree.getType() == TableGenParser.COMMA) {                        1000
                TGHash leftValue = evaluateExpression(expressionTree.getChild(0));
                TGHash rightValue = evaluateExpression(expressionTree.getChild(1));
                leftValue.concatenateToEnd(rightValue);
                result = leftValue;

                if (debug)
                        System.out.println("dbg: evaluateExpression-- concatenation result="+result.toString());
        }

            if (result == null) {                                                           1010
                if (debug)
                        System.out.println("dbg: null result in evaluateExpression at line " +
                                        expressionTree.getLine() + ". " + expressionTree.toStringTree ());
                //no exit because of return statements
            }
        }
        catch (TGTypeException e) {
                System.out.println(new TGTypeException(e.getMessage(),
                                expressionTree.getLine()).toString());
                System.exit(-1);                                                            1020
        }
        catch (TGSymbolNotFoundException e) {
                System.out.print(e.getMessage());
                System.out.println (" in evaluateExpression.");
                System.exit(-1);
        }

        return result;
}
                                                                                            1030
/**
 * Adds the numerical values in the hashes and throws the appropriate errors,
 * giving back a new TGHash object that is restricted to either NUM or STR type
 * @param x a TGHash to add to the other parameter
 * @param y a TGHash to add to the other parameter
```

```
 * @return a TGHash object containing the result value as its first and only
 *                      symbol that is restricted to hold only that symbol's type
 * @throws TGTypeException if either of the parameters has more than one element
 *                      or one of the values to add is a non-number type
 *                                                                              1040
 * ~!~Del
 */
public static TGHash add(TGHash x, TGHash y)
        throws TGTypeException
{
        TGHash result = new TGHash();
        result.setType(TGSymbol.TG_NUM_TYPE);

        if (x.getSymbolCount() != 1 || y.getSymbolCount() != 1)
                throw new TGTypeException("invalid symbol count in add operation");     1050

        if (!x.getFirstSymbol().isRuntimeNum() || !y.getFirstSymbol().isRuntimeNum())
                throw new TGTypeException("attempted addition includes a non-number value");

        /* both operands are integers – the result is also an integer */
        if (x.getFirstSymbol().isRuntimeInt() && y.getFirstSymbol().isRuntimeInt()) {
                int value = x.getFirstSymbol().getIntegerValue().intValue() +
                                        y.getFirstSymbol().getIntegerValue().intValue();
                result.addAsFirst(new TGSymbol("temp", new TGNum(value)));
        }                                                                       1060

        /* at least one operand is a double – the result is also a double */
        else {
                if (x.getFirstSymbol().getNumValue().isInt()) {
                        int xVal = x.getFirstSymbol().getIntegerValue().intValue();
                        double yVal = y.getFirstSymbol().getDoubleValue().doubleValue();
                        result.addAsFirst(new TGSymbol("temp", new TGNum(xVal+yVal)));
                }
                else {
                        double xVal = x.getFirstSymbol().getDoubleValue().doubleValue();        1070
                        if (y.getFirstSymbol().getNumValue().isInt()) {
                                int yVal = y.getFirstSymbol().getIntegerValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal+yVal)));
                        }
                        else {
                                double yVal = y.getFirstSymbol().getDoubleValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal+yVal)));
                        }
                }
        }                                                                       1080

        return result;
}

/**
 * Subtracts the numerical values in the hashes and throws the appropriate errors,
 * giving back a new TGHash object that is restricted to either NUM or STR type
 * @param x a TGHash to have the other parameter subtracted from it
 * @param y a TGHash to subtract from the other parameter
```

```
 * @return a TGHash object containing the result value as its first and only
 *                      symbol that is restricted to hold only that symbol's type
 * @throws TGTypeException if either of the parameters has more than one element
 *                      or one of the values to add is a non-number type
 */
public static TGHash subtract(TGHash x, TGHash y)
        throws TGTypeException
{
        TGHash result = new TGHash();
        result.setType(TGSymbol.TG_NUM_TYPE);

        if (x.getSymbolCount() != 1 || y.getSymbolCount() != 1)
                throw new TGTypeException("invalid symbol count in add operation");

        if (!x.getFirstSymbol().isRuntimeNum() || !y.getFirstSymbol().isRuntimeNum())
                throw new TGTypeException("attempted addition includes a non-number value");

        /* both operands are integers – the result is also an integer */
        if (x.getFirstSymbol().isRuntimeInt() && y.getFirstSymbol().isRuntimeInt()) {
                int value = x.getFirstSymbol().getIntegerValue().intValue() −
                                        y.getFirstSymbol().getIntegerValue().intValue();
                result.addAsFirst(new TGSymbol("temp", new TGNum(value)));
        }

        /* at least one operand is a double – the result is also a double */
        else {
                if (x.getFirstSymbol().getNumValue().isInt()) {
                        int xVal = x.getFirstSymbol().getIntegerValue().intValue();
                        double yVal = y.getFirstSymbol().getDoubleValue().doubleValue();
                        result.addAsFirst(new TGSymbol("temp", new TGNum(xVal−yVal)));
                }
                else {
                        double xVal = x.getFirstSymbol().getDoubleValue().doubleValue();
                        if (y.getFirstSymbol().getNumValue().isInt()) {
                                int yVal = y.getFirstSymbol().getIntegerValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal−yVal)));
                        }
                        else {
                                double yVal = y.getFirstSymbol().getDoubleValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal−yVal)));
                        }
                }
        }

        return result;
}

/**
 * Multiplies the numerical values in the hashes and throws the appropriate errors,
 * giving back a new TGHash object that is restricted to either NUM or STR type
 * @param x a TGHash multiplicand 1
 * @param y a TGHash multiplicand 2
 * @return a TGHash object containing the result value as its first and only
 *                      symbol that is restricted to hold only that symbol's type
```

```
 * @throws TGTypeException if either of the parameters has more than one element
 *                         or one of the values to add is a non-number type
 */
public static TGHash mult(TGHash x, TGHash y)
        throws TGTypeException
{
        TGHash result = new TGHash();                                                    1150
        result.setType(TGSymbol.TG_NUM_TYPE);

        if (x.getSymbolCount() != 1 || y.getSymbolCount() != 1)
                throw new TGTypeException("invalid symbol count in multiply operation");

        if (!x.getFirstSymbol().isRuntimeNum() || !y.getFirstSymbol().isRuntimeNum())
                throw new TGTypeException("attempted multiplication includes a non-number value");

        /* both operands are integers – the result is also an integer */
        if (x.getFirstSymbol().isRuntimeInt() && y.getFirstSymbol().isRuntimeInt()) {    1160
                int value = x.getFirstSymbol().getIntegerValue().intValue() *
                                        y.getFirstSymbol().getIntegerValue().intValue();
                result.addAsFirst(new TGSymbol("temp", new TGNum(value)));
        }

        /* at least one operand is a double – the result is also a double */
        else {
                if (x.getFirstSymbol().getNumValue().isInt()) {
                        int xVal = x.getFirstSymbol().getIntegerValue().intValue();
                        double yVal = y.getFirstSymbol().getDoubleValue().doubleValue();  1170
                        result.addAsFirst(new TGSymbol("temp", new TGNum(xVal*yVal)));
                }
                else {
                        double xVal = x.getFirstSymbol().getDoubleValue().doubleValue();
                        if (y.getFirstSymbol().getNumValue().isInt()) {
                                int yVal = y.getFirstSymbol().getIntegerValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal*yVal)));
                        }
                        else {
                                double yVal = y.getFirstSymbol().getDoubleValue();        1180
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal*yVal)));
                        }
                }
        }

        return result;
}

/**
 * Divides the numerical values in the hashes and throws the appropriate errors,    1190
 * giving back a new TGHash object that is restricted to either NUM or STR type
 * @param x a TGHash dividend
 * @param y a TGHash divisor
 * @return a TGHash object containing the result value as its first and only
 *                         symbol that is restricted to hold only that symbol's type
 * @throws TGTypeException if either of the parameters has more than one element
 *                         or one of the values to add is a non-number type
```

```
 */
public static TGHash divide(TGHash x, TGHash y)
        throws TGTypeException                                                  1200
{
        TGHash result = new TGHash();
        result.setType(TGSymbol.TG_NUM_TYPE);

        if (x.getSymbolCount() != 1 || y.getSymbolCount() != 1)
                throw new TGTypeException("invalid symbol count in divide operation");

        if (!x.getFirstSymbol().isRuntimeNum() || !y.getFirstSymbol().isRuntimeNum())
                throw new TGTypeException("attempted division includes a non-number value");
                                                                                1210
        // Division must return double if dividing two ints such as: 4 / 3
        if (x.getFirstSymbol().isRuntimeInt() && y.getFirstSymbol().isRuntimeInt()) {
                boolean remainder = true;
                if (x.getFirstSymbol().getIntegerValue().intValue() %
                    y.getFirstSymbol().getIntegerValue().intValue() > 0)
                        remainder = false;
                double value = (double)x.getFirstSymbol().getIntegerValue().intValue() /
                                (double)y.getFirstSymbol().getIntegerValue().intValue();
                /* If the arithmetic is returns int, return int, otherwise return double.
                 */                                                             1220
                if (remainder)
                        result.addAsFirst(new TGSymbol("temp", new TGNum((int)value)));
                else
                        result.addAsFirst(new TGSymbol("temp", new TGNum(value)));
        }

        /* at least one operand is a double – the result is also a double */
        else {
                if (x.getFirstSymbol().getNumValue().isInt()) {
                        int xVal = x.getFirstSymbol().getIntegerValue().intValue();     1230
                        double yVal = y.getFirstSymbol().getDoubleValue().doubleValue();
                        result.addAsFirst(new TGSymbol("temp", new TGNum(xVal/yVal)));
                }
                else {
                        double xVal = x.getFirstSymbol().getDoubleValue().doubleValue();
                        if (y.getFirstSymbol().getNumValue().isInt()) {
                                int yVal = y.getFirstSymbol().getIntegerValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal/yVal)));
                        }
                        else {                                                  1240
                                double yVal = y.getFirstSymbol().getDoubleValue();
                                result.addAsFirst(new TGSymbol("temp", new TGNum(xVal/yVal)));
                        }
                }
        }

        return result;
}

/**                                                                             1250
 * Compares the numerical values in the hashes and throws the appropriate errors,
```

```
 * giving back 1 (true) or 0 (false)
 * @param x a TGHash number 1
 * @param y a TGHash number 2
 * @return a TGHash object containing TGNum 1 or 0
 * @throws TGTypeException if either of the parameters has more than one element
 *                         or one of the values to add is a non-number type
 *
 * Daniel
 * ~!~Del – modified to allow string comparison as per the LRM                    1260
 */
public static TGHash compare(TGHash x, TGHash y, String type) throws TGTypeException {
    TGHash result = new TGHash();

    if (x.getSymbolCount() != 1 || y.getSymbolCount() != 1)
            throw new TGTypeException("invalid symbol count in comparison operation");

    result.setType(TGSymbol.TG_NUM_TYPE);

    /* string comparison */                                                        1270
    if (x.getFirstSymbol().getRuntimeType() == TGSymbol.TG_STR_TYPE &&
                y.getFirstSymbol().getRuntimeType() == TGSymbol.TG_STR_TYPE)
    {
        String xVal = x.getFirstSymbol().getStringValue();
                String yVal = y.getFirstSymbol().getStringValue();

                if ( (xVal.equals(yVal) && type.equals("==")) || (!xVal.equals(yVal) && type.equals("!=")))
                        result.addAsFirst(new TGSymbol("temp", new TGNum(1)));
                else
                        result.addAsFirst(new TGSymbol("temp", new TGNum(0)));                  1280
    }
    else {
        try {
                if (!x.getFirstSymbol().isRuntimeNum() || !y.getFirstSymbol().isRuntimeNum())
                throw new TGTypeException("invalid comparision: cannot compare strings and numbers");

                double left = x.getFirstSymbol().getDoubleValue().doubleValue();
                double right = y.getFirstSymbol().getDoubleValue().doubleValue();

                if (compare (left, right, type))                                        1290
                        result.addAsFirst(new TGSymbol("temp", new TGNum(1)));
                else
                        result.addAsFirst(new TGSymbol("temp", new TGNum(0)));
            } catch (TGTypeException e1) {
                int left = x.getFirstSymbol().getIntegerValue().intValue();
                int right = y.getFirstSymbol().getIntegerValue().intValue();

                if (compare (left, right, type))
                        result.addAsFirst(new TGSymbol("temp", new TGNum(1)));
                else                                                                    1300
                        result.addAsFirst(new TGSymbol("temp", new TGNum(0)));
            }
        }

    return result;
```

```
}

public static boolean compare (double left, double right, String type) {
        return (type.equals("==") && (left == right)) ||
                (type.equals("!=") && (left != right)) ||
                (type.equals("<") && (left < right)) ||
                (type.equals(">") && (left > right)) ||
                (type.equals("<=") && (left <= right)) ||
                (type.equals(">=") && (left >= right));
}

public static boolean compare (int left, int right, String type) {
        return (type.equals("==") && (left == right)) ||
                (type.equals("!=") && (left != right)) ||
                (type.equals("<") && (left < right)) ||
                (type.equals(">") && (left > right)) ||
                (type.equals("<=") && (left <= right)) ||
                (type.equals(">=") && (left >= right));
}

/**
 * Outputs the table data in tab-delimited format
 */
private void outputTable() {
        for(int i = 0; i <= maxRow; ++i)
        {
                int size = outputTable.get (i).size();
                for(int j = 0; j < size; ++j)
                {
                        String str = outputTable.get(i).get(j);

                        // Make java not print "nulls"
                        try {
                                str.length (); // this is a random op
                                               // can be anything
                        } catch (NullPointerException e) {
                                System.out.print("\t");
                                continue;
                        }

                        // Remove quotes from strings.
                        str = str.replaceFirst ("\"", "");
                        str = str.replaceAll ("\"$", "");

                        // Remove quotes and commas from between strings.
                        str = str.replaceAll ("\",[ ]*\"", " ");

                        // Remove escapes ("\")
                        str = str.replace ("\\", "");

                        System.out.print(str);

                        // only prints tabs between rows
                        // and not at the end of the table
```

```
                    if (j < size - 1)                                    1360
                        System.out.print("\t");
                }


                // only print new lines between rows
                // and not at the end of the table
                if (i < outputTable.size() - 1)
                System.out.print("\n");
            }
        }                                                                1370
}
```

8.2 Symbol Table

8.2.1 symtb/TGAbstractSymbolTable.java

This code builds our symbol table.
Signed: Del Slane

```
package symtb;

import java.util.*;
import org.antlr.runtime.tree.*;
import gram.*;
import excep.*;
import main.*;

public class TGAbstractSymbolTable {
    public boolean debug = TableGen.debug;                               10

    private CommonTree ast;
    private TGScope globalScope;

    private LinkedList<FunctionScopeEntry> functionList;

    public TGAbstractSymbolTable(CommonTree ast) {
        this.ast = ast;
        this.globalScope = new TGScope(null);
        this.functionList = new LinkedList<FunctionScopeEntry>();        20
    }

    public void printScopes() {
        if (debug)
            System.out.println("\nScopes:");
        printScopes(this.globalScope, "");
    }
```

```
private void printScopes(TGScope scope, String prefix) {
    if (debug)
            System.out.println(prefix + scope.toString());
    for (int i = 0; i < scope.getNumChildScopes(); i++)
            printScopes(scope.getChildScope(i), prefix + "\t");
}

private void printFunctionList() {
    for (int i = 0; i < functionList.size(); i++)
            if (debug)
                    System.out.println(functionList.get(i).toString());
}

/**
 * The main method called right after creation to actually build the data structure
 *
 * ~!~Del
 */
public void build() {
    try {
            this.buildFunctionList(ast);
            this.printFunctionList(); // debugging, implemented above
            this.buildScope(ast, globalScope);
    }
    catch (TGIllegalDeclarationException e) {
            System.out.println("Function declaration encountered outside global scope: " +
                            e.getMessage());
            System.exit(-1);
    }

    /* note that it is not necessary to manually pair up functions with the
     * just-created scopes because it is internal to buildScopes()
     */
}

/* a method that builds up data on functions since they can be declared anywhere
 * in global scope and are not necessarily forward-declared as with variables
 *
 * ~!~Del
 */
private void buildFunctionList(Tree ast) throws TGIllegalDeclarationException {
    /* function declarations must be at the global scope, so check all
     * first-level children of the tree.  If the child is not itself a function
     * definition, it's tree should not contain any function definitions.  If it
     * is, then its block's subtree should not have any
     */
    for (int i = 0; i < ast.getChildCount(); i++) {
            Tree currentChild = ast.getChild(i);

            if (currentChild.getType() == TableGenParser.FUNC_DEF) {
                    TGScope functionScope = new TGScope(null);

                    /* step1: register all function data with this.functionList */
                    String funcName = currentChild.getChild(0).getText();
```

```
                LinkedList<FunctionParameter> params = new LinkedList<FunctionParameter>();

                int j; // must be declared outside for final call to verifyNoFuncDefs
                for (j = 1; j < currentChild.getChildCount() &&
                                currentChild.getChild(j).getType() != TableGenParser.BLOCK; j++)
                {
                        Tree funcParam = currentChild.getChild(j);
                                                                                        90
                        /* case for a type-checked parameter */
                        if (funcParam.getChildCount() >= 2) {
                                int paramType = tgTypeStringToInt(funcParam.getChild(0).getText());
                                String paramName = funcParam.getChild(1).getText();

                                if (previousDeclarationOf(paramName, this.globalScope) == null)
                                        params.addLast(new FunctionParameter(paramName, paramType));
                                else
                                        throw new TGIllegalDeclarationException(paramName, currentChild.getLine());
                                                                                        100
                                functionScope.addSymbol(paramName, paramType);
                        }

                        /* parameter looking at does not have a specified type */
                        else {
                                params.addLast(new FunctionParameter(funcParam.getText()));
                                functionScope.addSymbol(funcParam.getText(), TGSymbol.NO_TYPE);
                        }
                }
                                                                                        110
                FunctionScopeEntry funcScopeEntry = new FunctionScopeEntry(funcName, params);
                funcScopeEntry.scope = functionScope;
                this.functionList.addLast(funcScopeEntry);

                /* step2: make sure the block doesn't contain any function definitions */
                verifyNoFunctionDefs(ast.getChild(ast.getChildCount() - 1));
            }

            /* child of global scope, not a function definition */
            else {                                                                      120
                if (ast.getChild(i) != null)
                        verifyNoFunctionDefs(ast.getChild(i));
            }
        }
}

/*
 * This method is called from buildFunctionList() and verifies that there are no
 * function definitions in the passed tree's subtrees
 *                                                                                      130
 * the passed tree should not be at the global scope
 *
 * ~!~Del
 */
private void verifyNoFunctionDefs(Tree tree)
                throws TGIllegalDeclarationException
```

```
{
    /* verify that the type of each node is NOT the "func" keyword */
    for (int i = 0; i < tree.getChildCount(); i++) {
            Tree child = tree.getChild(i);

            if (child.getType() == TableGenParser.FUNC_DEF) {
                    String functionName = child.getChild(0).getText();
                    int lineNumber = child.getLine();
                    throw new TGIllegalDeclarationException(functionName, lineNumber);
            }

            if (child.getChildCount() > 0)
                    verifyNoFunctionDefs(child);
    }
}

/*
 * A strictly internal method for recursively building up the abstract
 * symbol table. It must make sure to set the scopes of functions when they are
 * created.
 *
 * ~!~Del
 */
private void buildScope(Tree tree, TGScope currentScope) {
    boolean elseIsOk = false;

    if (debug)
            System.out.println("*dbg: SymbTb.buildScope-- tree="+tree.toStringTree());

    for (int i = 0; i < tree.getChildCount(); i++) {
            Tree child = tree.getChild(i);
            if (debug)
                    System.out.println("**dbg: SymbTb.buildScope-- child="+child.toStringTree());
            try {
                    /*
                     * do error checking on tree structure here – don't do it from a subtree
                     */
                    if (child.getType() == TableGenLexer.IF_KEYWORD)
                            elseIsOk = true;
                    else if (child.getType() == TableGenLexer.ELSEIF_KEYWORD ||
                                    child.getType() == TableGenLexer.ELSE_KEYWORD)
                    {
                            if (!elseIsOk)
                                    throw new TGIllegalElseException(child.getLine());
                    }
                    else
                            elseIsOk = false;

                    if (child.getType() == TableGenLexer.FUNC_DEF)
                            doFuncDef(child);
                    else if (child.getType() == TableGenLexer.ASSIGNMENT_OP)
                            doAssignmentOp(child, currentScope);
                    else if (child.getType() == TableGenLexer.PUT_OP)
                            doPutOp(child, currentScope);
```

```
                else if (child.getType() == TableGenParser.WHILE_KEYWORD)
                        doWhileLoop(child, currentScope);
                else if (child.getType() == TableGenParser.FOR_KEYWORD)
                        doForLoop(child, currentScope);
                else if (child.getType() == TableGenParser.FOREACH_KEYWORD)
                        doForeachLoop(child, currentScope);
                else if (child.getType() == TableGenParser.IF_KEYWORD ||
                                child.getType() == TableGenParser.ELSEIF_KEYWORD)
                        doIf(child, currentScope);
                else if (child.getType() == TableGenParser.ELSE_KEYWORD)
                        doElse(child, currentScope);
                else if (child.getType() == TableGenParser.FUNC_CALL)
                        doFuncCall(child, currentScope);

            }
            catch (TGUndefinedSymbolException e) {
                    System.out.println(e.getMessage());
                    System.exit(-1);
            }
            catch (TGIllegalDeclarationException e) {
                    System.out.println(e.getMessage());
                    System.exit(-1);
            }
            catch (TGMalformedFunctionCallException e) {
                    System.out.println(e.getMessage());
                    System.exit(-1);
            }
            catch (TGIllegalElseException e) {
                    System.out.println(e.getMessage());
                    System.exit(-1);
            }

    }
}

/*
 * Adds the result of a recursive call to buildScopes on the function's block
 * to the list of children scopes of currentScope and then update the entry in the
 * instance variable holding the list of function information
 *
 * ~!~Del
 */
private void doFuncDef(Tree funcDefTree) {
    String funcName = funcDefTree.getChild(0).getText();
            Tree funcBlock = funcDefTree.getChild(funcDefTree.getChildCount() - 1);

            FunctionScopeEntry[] funcEntries = new FunctionScopeEntry[this.functionList.size()];
            funcEntries = this.functionList.toArray(funcEntries);

            for (int j = 0; j < funcEntries.length; j++) {
                    if (funcEntries[j].name.equals(funcName)) {
                            buildScope(funcBlock, funcEntries[j].scope);
                            break;
                    }
```

```
            }
}

/*                                  ASSIGNMENT
     *
     * child1: TYPE_DECL -> if not empty, has a type declaration                                    250
     * child2: the identifier being given a value
     * child3: LIST_CONCAT -> for each child, is a value unless it is a ">", in which
     *                        case the child represents a hash key assignment – forces a hash type
     *
     * ~!~Del
     */
private void doAssignmentOp(Tree assignmentTree, TGScope parentScope)
                    throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{
    if (assignmentTree.getChild(0) == null || assignmentTree.getChild(0).getType() != TableGenLexer.TYPE_DECL) 260
                    System.err.println("bad value for first child of assignment op");

            /* gets the existing symbol table entry for the l-value in the assignment */
            TGHash declaredSymbol = previousDeclarationOf(assignmentTree.getChild(1).getText(), parentScope);

            /* error if attempting to redeclare a variable */
            if (declaredSymbol != null && assignmentTree.getChild(0).getChildCount() > 0) {
                    System.out.print("Attempted redeclaration of variable: ");
                    throw new TGUndefinedSymbolException(declaredSymbol.getName(), assignmentTree.getChild(1).getLine(
            }                                                                                    270

            /* assignmentTree.getChild(2) is the LIST_CONCAT that holds r-values */
            Tree rvalues = assignmentTree.getChild(2);
            validateTreeSymbols(rvalues, parentScope);

            /* add the symbol to the current scope if it has not already been declared */
            if (declaredSymbol == null) {
                    int type;
                    String variableName = assignmentTree.getChild(1).getText();
                                                                                                 280
                    /* the variable has a type declaration */
                    if (assignmentTree.getChild(0).getChildCount() > 0) {
                            type = tgTypeStringToInt(assignmentTree.getChild(0).getChild(0).getText());

                            parentScope.addSymbol(variableName, type);
                    }

                    /* no type declaration with the variable */
                    else {
                            type = TGSymbol.NO_TYPE;                                             290
                            parentScope.addSymbol(new TGHash(variableName));

                    }
                    //System.out.println("dbg: adding symbol '" + variableName + "' to current scope as type " + type);
            }
}

private void doPutOp(Tree putOpTree, TGScope parentScope)
```

```
                throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{                                                                                    300
    /* handle the expression on the left of the putop – it IS the expression subtree */
    validateTreeSymbols(putOpTree.getChild(0), parentScope);

    /* handle the coordinate – validate both expressions */
    validateTreeSymbols(putOpTree.getChild(1).getChild(0), parentScope);
    validateTreeSymbols(putOpTree.getChild(1).getChild(1), parentScope);

}

private void doWhileLoop(Tree whileTree, TGScope parentScope)                        310
                throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{
    /* validate all symbols in the expression */
    validateTreeSymbols(whileTree.getChild(0), parentScope);

    /* make a child scope and build it up for the while loop using a call to
     * buildScope()
     */
    TGScope whileScope = new TGScope(parentScope);
    buildScope(whileTree.getChild(1), whileScope);                                   320
}

/* ~!~Del */
private void doForLoop(Tree forLoopTree, TGScope parentScope)
                throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{
    /* since both expressions and assignment are allowed in the first/last parts of
     * the for loop declaration, we must make sure to add any new variables in the
     * assignment to the scope for the for loop before processing the body – this is
     * taken care of by doAssignmentOp()                                             330
     */
    TGScope forScope = new TGScope(parentScope);

    if (forLoopTree.getChild(0).getType() == TableGenParser.ASSIGNMENT_OP)
            doAssignmentOp(forLoopTree.getChild(0), forScope);

    /* pass the currentScope below to keep from the extra function call if passing the
     * for scope – I mean, we know it's empty . . .
     */
    else                                                                             340
            validateTreeSymbols(forLoopTree.getChild(0), parentScope);

    /* make sure the boolean expression has valid symbols */
    validateTreeSymbols(forLoopTree.getChild(1), forScope);

    /* does this 'if' case have a problem with defining a new variable in the final statement?
     * is that a runtime exception if so?
     */
    if (forLoopTree.getChild(2).getType() == TableGenParser.ASSIGNMENT_OP)
            doAssignmentOp(forLoopTree.getChild(2), forScope);                       350
    else
            validateTreeSymbols(forLoopTree.getChild(2), forScope);
```

```
    buildScope(forLoopTree.getChild(3), forScope);
}

/* ~!~Del */
private void doForeachLoop(Tree foreachLoopTree, TGScope parentScope)
                    throws TGUndefinedSymbolException, TGIllegalDeclarationException
{                                                                                    360
    TGScope foreachScope = new TGScope(parentScope);
    int type = TGSymbol.NO_TYPE;

    Tree iteratorChild = foreachLoopTree.getChild(1);
    String iteratorName = iteratorChild.getText();
    TGHash previouslyDeclared = previousDeclarationOf(iteratorName, parentScope);

    /* iterator must be a new variable – throw error otherwise */
    if (previouslyDeclared != null)
            throw new TGIllegalDeclarationException(previouslyDeclared.getName(), iteratorChild.getLine()); 370

    /* handle typed iterator */
    if (foreachLoopTree.getChild(0).getChildCount() > 0)
            type = tgTypeStringToInt(foreachLoopTree.getChild(0).getChild(0).getText());

    foreachScope.addSymbol(iteratorName, type);

    buildScope(foreachLoopTree.getChild(3), foreachScope);

}                                                                                    380

private void doIf(Tree ifTree, TGScope parentScope)
                    throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{
    TGScope ifScope = new TGScope(parentScope);
    validateTreeSymbols(ifTree.getChild(0), parentScope); // expression to evaluate
    buildScope(ifTree.getChild(1), ifScope); // build the scope for the block
}

private void doElse(Tree elseTree, TGScope parentScope)                              390
                    throws TGUndefinedSymbolException
{
    TGScope elseScope = new TGScope(parentScope);
    buildScope(elseTree.getChild(0), elseScope);
}

/**
 * A method that will validate a function call given a FUNC_CALL AST node
 * param funcCallTree the FUNC_CALL AST node corresponding to the function call
 * param parentScope the scope from which this function was called                   400
 * throws TGUndefinedSymbolException when a named parameter is not found in the
 *                    list of valid parameter names for the function, or if the function
 *                    was not defined
 * throws TGMalformedFunctionCallException when the number of parameters is incorrect
 *
 * ~!~Del
```

```
 */
private void doFuncCall(Tree funcCallTree, TGScope parentScope)
                throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{                                                                                    410
    String funcName = funcCallTree.getChild(0).getText();
    if (debug)
        System.out.println (funcName + ": " + funcCallTree.toStringTree ());

    // If reversed function, do something special.
    if (funcName.matches ("^_.+")) {
        return;
    }

    int callParamCount = funcCallTree.getChildCount() - 1; // don't count the name as a parameter   420
    FunctionScopeEntry function = getFunctionScopeEntry(funcName);

    if (function == null) {
        throw new TGUndefinedSymbolException(funcName, funcCallTree.getLine());
    }

    if (callParamCount != function.getParamCount())
        throw new TGMalformedFunctionCallException(funcName, funcCallTree.getLine());

    boolean sawNamedParam = false;                                                   430
    boolean sawUnnamedParam = false;

    /* check each parameter for valid values */
    for (int j = 1; j < funcCallTree.getChildCount(); j++) {
        Tree param = funcCallTree.getChild(j);

        /* check named parameter */
        if (param.getText().equals(":")) {
            String paramName = param.getChild(0).getText();
            boolean isValidParam = isValidFuncParam(funcName, paramName);            440

            if (!isValidParam) {
                System.out.println("Unable to match function '" + funcName +
                                            " with parameter " + paramName + ": ");
                throw new TGUndefinedSymbolException(paramName, param.getChild(0).getLine());
            }
            sawNamedParam = true;
        }

        /* check literal or identifier as value-only parameter */                    450
        else {
            if (!isValidSymbol(param.getText(), parentScope))
                throw new TGUndefinedSymbolException(param.getText(), param.getLine());
            sawUnnamedParam = true;
        }

        /* cannot allow mixing for interpreting issues that follow */
        if (sawNamedParam && sawUnnamedParam)
            throw new TGMalformedFunctionCallException("Illegal mixing of named/unnamed parameters "+
                        "in call to function '" + funcName + "'", funcCallTree.getLine());  460
```

```
    }
}

/**
 * Returns the TGSymbol object corresponding to the previous declaration of the given
 * variable or function name name or null if the name is not already defined
 * param name The name of the identifier to search for
 * return The corresponding TGSymbol object if a match was found, otherwise null. If
 *                  the symbol found was a function, the name field is the matched name but the
 *                  type is NO_TYPE. It is also a temporary returned value, not one that is
 *                  actually held in a given scope.
 *
 * ~!~Del
 */
private TGHash previousDeclarationOf(String name, TGScope scope) {
    TGHash[] symbols; // symbols for the current scope

    for (; scope != null; scope = scope.getParentScope()) {
            symbols = scope.getSymbols();
            for (int i = 0; i < symbols.length; i++) {
                    if (symbols[i].getName().equals(name))
                            return symbols[i];
            }
    }

    /* check the list of functions */
    FunctionScopeEntry[] funcs = new FunctionScopeEntry[this.functionList.size()];
    funcs = this.functionList.toArray(funcs);
    for (int i = 0; i < funcs.length; i++) {
            if (funcs[i].name.equals(name))
                    return new TGHash(name); /* returns a temporary symbol with no type for a function */
    }

    return null;
}

/**
 * Makes sure that all values in the given tree have been previously declared, ignoring operators
 * and including function calls and their parameters. This method is primarily used in static-
 * semantic analysis checks of statements in the buildScopes() method
 * param tree The subtree of the AST to evaluate
 * throws a TGUndefinedSymbolException if a tree symbol has not been previously defined
 *
 * ~!~Del
 */
private void validateTreeSymbols(Tree tree, TGScope currentScope)
                    throws TGUndefinedSymbolException, TGMalformedFunctionCallException
{
    /* the current node is a tree with children */
            if (tree.getChildCount() >= 1) {

                    if (tree.getType() == TableGenParser.FUNC_CALL)
                            doFuncCall(tree, currentScope);
```

```
                /* user must keep track of making sure list keys are valid */
                else if (tree.getType() == TableGenParser.HASH_ASSIGN);
                else if (tree.getType() == TableGenParser.DECIMAL_POINT)
                        validateTreeSymbols(tree.getChild(0), currentScope);

                /* trees with children but that are not function call trees – validate children */
                else {
                        for (int i = 0; i < tree.getChildCount(); i++)
                                validateTreeSymbols(tree.getChild(i), currentScope);
                }
        }

        /* the current node is not a tree with children – must be an identifier, literal, operator,
         * function call name, or unnamed function call parameter (named function call parameter
         * handled in the tree case)
         */
        else if (!isValidSymbol(tree.getText(), currentScope))
                        throw new TGUndefinedSymbolException(tree.getText(), tree.getLine());

}

/* ~!~Del */
private boolean isValidFuncParam(String funcName, String paramName) {
    boolean paramFound = false;
        FunctionScopeEntry[] funcs = new FunctionScopeEntry[this.functionList.size()];
        funcs = this.functionList.toArray(funcs);

        for (int k = 0; k < funcs.length; k++) {

                /* we found the function to match the named parameter with */
                if (funcs[k].name.equals(funcName)) {
                        FunctionParameter[] params = new FunctionParameter[funcs[k].params.size()];
                        params = funcs[k].params.toArray(params);

                        /* look through the parameters for a match */
                        for (int l = 0; l < params.length; l++) {
                                if (params[l].name.equals(paramName))
                                        paramFound = true;
                        }
                }

        }

        return paramFound;
}

/* Gets the type of the r-values in assignment ~!~ Del */
public static int tgTypeFromListConcatNode(Tree listConcatNode) {
    int type;

    try {
                Integer.parseInt(listConcatNode.getChild(0).getText());
                type = TGSymbol.TG_NUM_TYPE;
        }
```

```
            catch (NumberFormatException e) {
                try {
                    Double.parseDouble(listConcatNode.getChild(0).getText());
                    type = TGSymbol.TG_NUM_TYPE;
                }
                catch (NumberFormatException ex) {
                    type = TGSymbol.TG_STR_TYPE;
                }
            }

    return type;
}

/* ~!~Del */
public boolean isValidSymbol(String str, TGScope scope) {
    if (
                isStringLiteral(str) ||
                isNumberLiteral(str) ||
                isOperator(str)
       )
            return true;

    TGHash declared = previousDeclarationOf(str, scope);
            if (declared != null)
                return true;

    return false;
}

/* ~!~Del */
public static boolean isNumberLiteral(String str) {
    try {
            Integer.parseInt(str);
            return true;
    }
    catch (NumberFormatException e) {
            try {
                Double.parseDouble(str);
                return true;
            }
            catch (NumberFormatException ex) {
                return false;
            }
    }
}

public static boolean isStringLiteral(String str) {
    return str.charAt(0) == '"';
}

public static boolean isOperator(String str) {
    if (
            str.equals("+") ||
            str.equals("-") ||
```

```
            str.equals("*") ||
            str.equals("/") ||
            str.equals("&&") ||
            str.equals("||") ||
            str.equals("==") ||
            str.equals("->") ||
            str.equals("=") ||
            str.equals("<") ||
            str.equals(">") ||
            str.equals("<=") ||
            str.equals(">=") ||
            str.equals("!=")
        )
        return true;


    return false;
}

/**
 * This method takes in a string representation of a TG primitive type such as "num"
 * and returns the corresponding value from the TGSymbol class
 * param typeDeclaration The string representation of the type declaration
 * return The integer value corresponding to the type as in the TGSymbol class
 *                          (< 0 for invalid type)
 *
 * ~!~Del
 */
public static int tgTypeStringToInt(String typeDeclaration) {
    if (typeDeclaration.equals("num"))
            return TGSymbol.TG_NUM_TYPE;
    else if (typeDeclaration.equals("str"))
            return TGSymbol.TG_STR_TYPE;
    else
            return TGSymbol.NO_TYPE;
}

public TGScope getGlobalScope() {
    return this.globalScope;
}

/**
 * Gets the TGScope representing the scope of the specified function. Note that
 * all internal scopes such as in for loops and if blocks are children of the
 * returned scope.
 * param funcName The name of the function to get the scope of
 * return The TGScope of the specified function with the specified name or
 *                  null for not found
 *
 * ~!~Del
 */
public TGScope getFunctionScope(String funcName) {
    FunctionScopeEntry[] funcEntries = new FunctionScopeEntry[this.functionList.size()];
    funcEntries = this.functionList.toArray(funcEntries);
```

```
        for (int i = 0; i < funcEntries.length; i++) {
                if (funcEntries[i].name.equals(funcName))
                        return funcEntries[i].scope;
        }

        return null;
    }

    /**
     * Returns the function scope entry corresponding to the given name
     * param functionName
     * return
     */
    private FunctionScopeEntry getFunctionScopeEntry(String functionName) {
        FunctionScopeEntry[] funcScopeEntries = new FunctionScopeEntry[this.functionList.size()];
        funcScopeEntries = this.functionList.toArray(funcScopeEntries);

        for (int i = 0; i < funcScopeEntries.length; i++) {
                if (funcScopeEntries[i].name.equals(functionName))
                        return funcScopeEntries[i];
        }

        return null;
    }
}


/* ~!~Del */
class FunctionScopeEntry {
        String name;
        LinkedList<FunctionParameter> params;
        TGScope scope;

        /* this constructor is so that an initial pass of the tree can be made to validate names of
         * functions during tree walking since nothing is said about the declaration order
         */
        public FunctionScopeEntry(String name) {
                this.name = name;
        }

        public FunctionScopeEntry(String name, LinkedList<FunctionParameter> parameters) {
                this.name = name;
                this.params = parameters;
        }

        public String toString() {
                String result = "Function Name: " + this.name;
                result += "\n\t";

                for (int i = 0; i < params.size(); i++) {
                        FunctionParameter p = params.get(i);
                        result += "param " + i + ": type=" + p.type + ", name=" + p.name + "\n\t";
                }
```

```
                result += "\n";
                return result;
        }

        public int getParamCount() {
                return this.params.size();
        }
}

/* ~!~Del */
class FunctionParameter {
        String name;
        int type; /* types as defined in TGSymbol.java */

        public FunctionParameter(String name) {
                this.name = name;
                this.type = TGSymbol.NO_TYPE;
        }

        public FunctionParameter(String name, int type) {
                this.name = name;
                this.type = type;
        }
}
```

8.2.2 symtb/TGHash.java

This file hold code for our hash-key feature.
Signed: Del Slane

```
package symtb;

import excep.*;

import java.util.*;


/**
 * Things to note about the TGHash class:
 *
 * The keys of the elements are the names of the TGSymbols in the linked list
 *
 * If two keys of the hash are the same, the first one is returned
 */
public class TGHash implements Cloneable{
   private LinkedList<TGSymbol> elements;
   private int type;
   private String name;
```

```
public TGHash() {
        this("", TGSymbol.NO_TYPE);
}

public TGHash(String name) {
        this(name, TGSymbol.NO_TYPE);
}

/**
 * Creates a new TGHash that is restricted to the given type
 * @param type One of the types defined in TGSymbol
 */
public TGHash(String name, int type) {
        this.elements = new LinkedList<TGSymbol>();
        this.type = type;
        this.name = name;
}

public int getType() {
        return this.type;
}

public String getName() {
        return this.name;
}

public boolean hasType() {
        return this.type != TGSymbol.NO_TYPE;
}

/**
 * Restricts the types that can be in the current hash, deletes all values
 * currently in the hash
 * @param type One of the types defined in TGSymbol
 */
public void setType(int type) {
        this.type = type;
        this.elements = new LinkedList<TGSymbol>();
}

/**
 * Adds a new element to the end of the hash
 * @param symbol The element to add
 */
public void addAsLast(TGSymbol symbol)
            throws TGTypeException
{
        if (this.type != TGSymbol.NO_TYPE && symbol.getRuntimeType() != this.type)
                throw new TGTypeException("attempted addition of type " + symbol.getType() +
                                " to a hash of type " + this.type);
        this.elements.addLast(symbol);
}

/**
```

```
 * Adds a new element to the front of the hash
 * @param symbol The element to add
 */
public void addAsFirst(TGSymbol symbol)
        throws TGTypeException
{
    if (this.type != TGSymbol.NO_TYPE && symbol.getRuntimeType() != this.type)
        throw new TGTypeException("attempted addition of type " + symbol.getType() +
                " to a hash of type " + this.type);
    this.elements.addFirst(symbol);
}

/**
 * Concatenates the values from the passed TGHash onto the end of the list
 * of elements for the TGHash on which the method is called
 * @param tail the other TGHash whose elements to add as the new tail of
 *             the current TGHash's elements
 */
public void concatenateToEnd(TGHash tail) {
    TGSymbol[] symbolsToAdd = new TGSymbol[tail.getSymbolCount()];
    symbolsToAdd = tail.getSymbols();
    for (int i = 0; i < symbolsToAdd.length; i++)
        this.elements.addLast(symbolsToAdd[i]);
}

/**
 * Used to see if the hash is empty
 * @return Returns true if the hash does not contain any elements
 */
public boolean isEmpty() {
    return this.elements.isEmpty();
}

/**
 * Used to get the last element of hash
 * @return Returns the last element of the hash
 */
public TGHash getTail () {
    TGSymbol el = this.elements.getLast ();
    TGHash ret = new TGHash ();

    try {
        ret.addAsFirst (el);
    } catch (TGTypeException e) {
        System.out.println ("Failed to get last element of list.");
    }

    return ret;
}

/**
 * Used to get the length of the hash
 * @return Return the length of the hash
 */
```

```
public TGHash getLength () {
    int el = this.elements.size ();
    TGHash num = new TGHash ();

    try {
        num.setValue (el);
    } catch (TGTypeException e) {
        System.out.println ("Failure when trying to get length of variable.");
    }

    return num;
}
/**
 * Allows array-like accesses of a TGHash by getting the symbol given a valid index
 * @param index The index of the element sought
 * @return The TGSymbol at the given index of the hash
 */
public TGSymbol getSymbol(int index) {
    return this.elements.get(index);
}

public TGSymbol getFirstSymbol() {
    return this.elements.getFirst();
}

/**
 * Gets the TGSymbol of the specified name
 * @param name The name of the symbol to find
 * @return Returns the first TGSymbol object with the specified name and
 *                  null for not found
 */
public TGSymbol getElement(String hashKey)
        throws TGSymbolNotFoundException
{
    for (int i = 0; i < this.elements.size(); i++) {
        TGSymbol current = this.elements.get(i);
        if (current.getKey().equals(hashKey))
            return current;
    }

    throw new TGSymbolNotFoundException(hashKey);
}

/**
 * Gets the elements of the hash as an array of TGSymbols
 * @return The elements of the hash as an array
 */
public TGSymbol[] getSymbols() {
    TGSymbol[] result = new TGSymbol[this.elements.size()];
    return this.elements.toArray(result);
}

public int getSymbolCount() {
    return this.elements.size();
```

```
}

public void setValue(Integer i)
        throws TGTypeException
{
    if (this.type != TGSymbol.TG_NUM_TYPE && this.type != TGSymbol.NO_TYPE)
            throw new TGTypeException(this.name);
    this.elements = new LinkedList<TGSymbol>();
    this.elements.addFirst(new TGSymbol(new TGNum(i)));
}

public void setValue(Double d)
        throws TGTypeException
{
    if (this.type != TGSymbol.TG_NUM_TYPE && this.type != TGSymbol.NO_TYPE)
        throw new TGTypeException(this.name);
    this.elements = new LinkedList<TGSymbol>();
    this.elements.addFirst(new TGSymbol(new TGNum(d)));
}

public void setValue(String s)
        throws TGTypeException
{
    if (this.type != TGSymbol.TG_STR_TYPE && this.type != TGSymbol.NO_TYPE)
        throw new TGTypeException(this.name);
    this.elements = new LinkedList<TGSymbol>();
    this.elements.addFirst(new TGSymbol(s));
}

/**
 * Copies the values from the parameter into the hash and checks all types
 * as necessary
 * @param h the hash values to put into this hash
 * @throws TGTypeException if there is a type constraint on this hash and at least
 * one element of the parameter hash does not match that type constraint
 */
public void setValue(TGHash h)
        throws TGTypeException
{
    if (this.type == TGSymbol.NO_TYPE) {
        this.elements = new LinkedList<TGSymbol>();
        TGSymbol[] symbols = h.getSymbols();

        for (int i = 0; i < symbols.length; i++)
            this.elements.addLast(symbols[i]);
    }
    else {
        LinkedList<TGSymbol> potentialElements = new LinkedList<TGSymbol>();
        TGSymbol[] symbols = h.getSymbols();

        for (int i = 0; i < symbols.length; i++) {
            if (symbols[i].getType() != TGSymbol.NO_TYPE &&
                    symbols[i].getRuntimeType() != this.type)
                throw new TGTypeException(this.name);
```

```
                        else
                                potentialElements.addLast(symbols[i]);
                }

                this.elements = potentialElements;
        }
}

public void setName(String newName) {
        this.name = newName;
}

/**
 * Erases the values in the TGHash
 */
public void zero() {
        this.elements = new LinkedList<TGSymbol>();
}

/**
 * Takes in a string representation of a non-list hash value and converts it into
 * its TGHash representation
 * @param name the name for the TGHash to be returned
 * @param value the string representation of the value
 * @return the corresponding TGHash representation of the value
 * @throws TGTypeException if the provided string representation of the value cannot
 *                  be successfully parsed
 */
public static TGHash tgHashFromString(String name, String value)
            throws TGTypeException
{
        TGHash result = null;

        try {
                Integer i = Integer.parseInt(value);
                result = new TGHash(name, TGSymbol.TG_NUM_TYPE);
                result.addAsFirst(new TGSymbol(new TGNum(i)));
        } catch (NumberFormatException e) {
                try {
                        Double d = Double.parseDouble(value);
                        result = new TGHash(name, TGSymbol.TG_NUM_TYPE);
                        result.addAsFirst(new TGSymbol(new TGNum(d)));
                }
                catch (NumberFormatException ex) {
                        result = new TGHash(name, TGSymbol.TG_STR_TYPE);
                        result.addAsFirst(new TGSymbol(name, value));
                }

        }

        if (result != null)
                result.setName(name);

        return result;
```

```
    }

    public String toString() {
        String result = "";
        TGSymbol[] symbols = new TGSymbol[this.elements.size()];
        symbols = this.elements.toArray(symbols);
        for (int i = 0; i < symbols.length; i++) {
            result += symbols[i].toString();
            // Separate lists by commas
            if (!(i == (symbols.length - 1)))
                result += ",";
        }
        return result;
    }

    public TGHash clone() {
        TGHash clone = new TGHash(this.name, this.type);
        TGSymbol[] thisSymbols = this.getSymbols();

        try {
            for (int i = 0; i < thisSymbols.length; i++) {
                if (thisSymbols[i].isRuntimeInt())
                    clone.addAsLast(new TGSymbol(new TGNum(thisSymbols[i].getIntegerValue().intValue())));
                else if (thisSymbols[i].isRuntimeDouble())
                    clone.addAsLast(new TGSymbol(new TGNum(thisSymbols[i].getDoubleValue().doubleValue())));
                else // COPIES the value, not just passes it
                    clone.addAsLast(new TGSymbol(thisSymbols[i].getStringValue().substring(0)));
            }
        }
        catch (TGTypeException e) {
            System.out.println("Compiler error in TGHash.clone");
            System.exit(1);
        }

        return clone;
    }

}
```

8.2.3 symtb/TGNum.java

This file hold code for handling numbers.
Signed: Del Slane

```
package symtb;

public class TGNum {

    public static final int INT = 1;
    public static final int DOUBLE = 2;
```

```
private int type;
private Object value;
                                                                    10

public TGNum(Integer i){
    type = INT;
    value = i;
}

public TGNum(Double d){
    type = DOUBLE;
    value = d;
}                                                                   20

public TGNum(int i) {
    type = INT;
    value = new Integer(i);
}

public TGNum(double d) {
    type = DOUBLE;
    value = new Double(d);
}                                                                   30

public boolean isInt(){
    return (type == INT);
}

public boolean isDouble(){
    return (type == DOUBLE);
}

public int getIntValue(){                                           40
    return ((Integer)value).intValue();
}

public double getDoubleValue(){
    return ((Double)value).doubleValue();
}

public void setValue(int i){
    type = INT;
    value = new Integer(i);                                         50
}

public void setValue(double d){
    type = DOUBLE;
    value = new Double(d);
}

public String toString() {
    if (this.type == DOUBLE)
            return ((Double)(this.value)).toString();               60
```

```
        else
                return ((Integer)(this.value)).toString();
    }
}
```

8.2.4 symtb/TGScope.java

This file hold code for handling scopes.
Signed: Del Slane

```
package symtb;

import java.util.LinkedList;
import java.util.ListIterator;

public class TGScope {
    public static final TGScope NULL_SCOPE = null;

    private TGScope parentScope;
    private LinkedList<TGScope> childrenScopes;
    private LinkedList<TGHash> symbols;
    private int childIterator;

    public TGScope(TGScope parentScope){
        this.parentScope = parentScope;
        if (parentScope != null)
                parentScope.addToChildrenScopes(this);
        this.childrenScopes = new LinkedList<TGScope>();
        this.symbols = new LinkedList<TGHash>();
        this.childIterator = 0;
    }

    public TGScope getNextChildScope(){
        if (childIterator == this.childrenScopes.size())
                childIterator = 0;
        return childrenScopes.get(childIterator++);
    }

    public void resetScopeIterator() {
        childIterator = 0;
    }

    public LinkedList<TGScope> getChildrenScopes() {
        return childrenScopes;
    }

    public int getNumChildScopes(){
        return this.childrenScopes.size();
    }
```

```
public int getNumSymbols(){
    return this.symbols.size();
}

public TGHash getSymbol(int i){
    return symbols.get(i);
}

public TGScope getChildScope(int i){
    return childrenScopes.get(i);
}

public TGScope getParentScope(){
    return parentScope;
}

public int getSymbolType(String name){
    for (int i = 0; i < symbols.size(); i++){
        if (symbols.get(i).getName().equals(name))
            return symbols.get(i).getType();
    }
    return 0;
}

public boolean containsSymbol(String name){
    for (int i = 0; i < symbols.size(); i++){
        if (symbols.get(i).getName().equals(name))
            return true;
    }
    return false;
}

/**
 * Adds the given scope to the list of children for the scope called on
 * and also makes the scope called on the parent of the scope passed as
 * a parameter
 * @param s the scope to add to the list of children
 */
public void addToChildrenScopes(TGScope s) {
    this.childrenScopes.addLast(s);
    s.setParentScope(this);
}

public void addSymbol(TGHash s) {
    this.symbols.add(s);
}

/**
 * Makes a new symbol of the specified type and adds it to the scope
 * @param name the name of the symbol to add
 * @param type the type of the symbol to add as declared in TGSymbol
 */
public void addSymbol(String name, int type) {
    this.addSymbol(new TGHash(name, type));
```

```
    }

    public TGHash[] getSymbols() {
        TGHash[] result = new TGHash[symbols.size()];
        result = symbols.toArray(result);
        return result;
    }

    public String toString() {
        int i;
        String result = "";
        TGHash[] symbols = new TGHash[this.symbols.size()];
        symbols = this.symbols.toArray(symbols);

        if (symbols.length > 0) {
                for (i = 0; i < symbols.length - 1; i++)
                        result += symbols[i].getName() + ", ";
                result += symbols[i].getName();
        }

        return result;
    }

    void setParentScope(TGScope s) {
        this.parentScope = s;
    }
}
```

8.2.5 symtb/TGSymbol.java

This file hold code for symbols.
Signed: Del Slane

```
package symtb;

import excep.*;

public class TGSymbol {
        /* all valid types must be >= 0 for testing purposes */
        public static final int NO_TYPE = -1;
        public static final int TG_NUM_TYPE = 0;
        public static final int TG_STR_TYPE = 1;

    private String hashKey;
    private Object value;
    private int type;
    private int runtimeType;

    public TGSymbol(String hashKey, TGNum value) {
        this.hashKey = hashKey;
```

```
        this.value = value;
        this.type = NO_TYPE;
        this.runtimeType = TG_NUM_TYPE;
    }

    public TGSymbol(String hashKey, String value) {
        this.hashKey = hashKey;
        this.value = value;
        this.type = NO_TYPE;
        this.runtimeType = TG_STR_TYPE;
    }

    public TGSymbol(TGNum value) {
        this.hashKey = "";
        this.value = value;
        this.type = NO_TYPE;
        this.runtimeType = TG_NUM_TYPE;
    }

    public TGSymbol(String value) {
        this.hashKey = "";
        this.value = value;
        this.type = NO_TYPE;
        this.runtimeType = TG_STR_TYPE;
    }

    public TGSymbol(int i) {
        this.hashKey = "";
        this.type = NO_TYPE;
        this.runtimeType = TG_NUM_TYPE;
        this.value = new TGNum(i);
    }

    public TGSymbol(double d) {
        this.hashKey = "";
        this.type = NO_TYPE;
        this.runtimeType = TG_NUM_TYPE;
        this.value = new TGNum(d);
    }

    /**
     * Used to find the type of the symbol; knowing what type of symbol
     * is stored in the symbol is necessary before getting its value so the
     * appropriate method can be called
     * @return The type of the symbol: TG_NUM_TYPE, TG_STR_TYPE, or TG_HASH_TYPE
     */
    public int getType(){
        return type;
    }

    public boolean hasType() {
        return this.type == NO_TYPE;
    }
```

```
public boolean isNum() {
    return this.type == TG_NUM_TYPE;
}

public boolean isInt() {
    return this.type == TG_NUM_TYPE && ((TGNum)this.value).isInt();
}

public boolean isDouble() {                                                    80
    return this.type == TG_NUM_TYPE && ((TGNum)this.value).isDouble();
}

public boolean isString() {
    return this.type == TG_STR_TYPE;
}

public boolean isRuntimeNum() {
    return this.runtimeType == TG_NUM_TYPE;
}                                                                              90

public boolean isRuntimeInt() {
    return this.runtimeType == TG_NUM_TYPE && ((TGNum)this.value).isInt();
}

public boolean isRuntimeDouble() {
    return this.runtimeType == TG_NUM_TYPE && ((TGNum)this.value).isDouble();
}

public boolean isRuntimeString() {                                             100
    return this.runtimeType == TG_STR_TYPE;
}

public TGNum getNumValue()
        throws TGTypeException
{
    if (this.runtimeType != TG_NUM_TYPE)
        throw new TGTypeException("request for a num value for the non-num hash entry '" +
                this.hashKey + "'");
    return (TGNum)value;                                                       110
}

public Integer getIntegerValue()
        throws TGTypeException
{
    if (this.runtimeType != TG_NUM_TYPE || !((TGNum)value).isInt())
        throw new TGTypeException("request for an integer value for the non-integer hash entry '" +
                this.hashKey + "'");
    return ((TGNum)value).getIntValue();
}                                                                              120

public Double getDoubleValue()
        throws TGTypeException
{
    if (this.runtimeType != TG_NUM_TYPE || !((TGNum)value).isDouble())
```

```
            throw new TGTypeException("request for a double value for the non-double hash entry '" +
                        this.hashKey + "'");
    return ((TGNum)value).getDoubleValue();
}
                                                                                    130
public String getStringValue()
            throws TGTypeException
{
    if (this.runtimeType != TG_STR_TYPE)
            throw new TGTypeException("request for a string value for the non-string hash entry '" +
                        this.hashKey + "'");
    return (String)value;
}

public String getKey(){                                                             140
    return hashKey;
}

public void setKey(String newKey) {
    this.hashKey = newKey;
}

public void setValue(int i)
            throws TGTypeException
{                                                                                   150
    if (this.type != NO_TYPE && this.type != TG_NUM_TYPE)
            throw new TGTypeException("attempted assigment of an integer to the non-integer hash entry '" +
                        this.hashKey + "'");
    this.value = new TGNum(i);
}

public void setValue(double d)
            throws TGTypeException
{
    if (this.type != NO_TYPE && this.type != TG_NUM_TYPE)                           160
            throw new TGTypeException("attempted assignment of a double value to the non-double hash entry '"
                        this.hashKey + "'");
    this.value = new TGNum(d);
}

public void setValue(String s)
            throws TGTypeException
{
    if (this.type != NO_TYPE && this.type != TG_STR_TYPE)
            throw new TGTypeException("attempted assignment of a string value to the non-string hash entry '"
                        this.hashKey + "'");
    this.value = s;
}

/**
 * Gets the runtime type of the symbol
 * @return the integer value corresponding to type flags in this class:
 *                  TG_NUM_TYPE, or TG_STR_TYPE
 */
```

```
    public int getRuntimeType() {
        return this.runtimeType;
    }

    /**
     * Sets the runtime type of the TGSymbol
     * @param type should be one of the public constants in TGSymbol (ie. TG_NUM_TYPE)
     */
    public void setRuntimeType(int type) {
        this.runtimeType = type;
    }

    public String toString() {
        if (this.value == null)
                return "";
        if (this.runtimeType == TG_NUM_TYPE)
                return ((TGNum)(this.value)).toString();
        return (String)this.value;
    }

}
```

8.3 Exceptions

All exceptions in the excep folder.
Signed: Del Slane

8.3.1 excep/TGTypeException.java

```
package excep;

public class TGTypeException extends Exception {
        public TGTypeException(String message) {
                super("TGTypeException: " + message);
        }

        public TGTypeException(String message, int lineNumber) {
                super("TGTypeException: " + message + " on line " + lineNumber);
        }
}
```

8.3.2 excep/TGMissingReturnValueException.java

```
package excep;
```

```
public class TGMissingReturnValueException extends RuntimeException {

        public TGMissingReturnValueException(String functionName, int lineNumber) {
                super("Call to function '" + functionName +"' that requires a return value " +
                                "where none is specified on line " + lineNumber);
        }
}
```

8.3.3 excep/TGUndefinedSymbolException.java

```
package excep;

public class TGUndefinedSymbolException extends Exception {
        public TGUndefinedSymbolException(String symbolName, int lineNumber) {
                super("Reference to undefined symbol '" + symbolName + "' on line " + lineNumber);
        }
}
```

8.3.4 excep/TGIllegalDeclarationException.java

```
package excep;

public class TGIllegalDeclarationException extends Exception {
        public TGIllegalDeclarationException(String symbol, int lineNumber) {
                super("Illegal declaration of '" + symbol + "' on line " + lineNumber);
        }
}
```

8.3.5 excep/TGMalformedFunctionCallException.java

```
package excep;

public class TGMalformedFunctionCallException extends Exception {
        public TGMalformedFunctionCallException(String message, int lineNumber) {
                super("Malformed function call: " + message + " on line " + lineNumber);
        }
}
```

8.3.6 excep/TGIllegalElseException.java

```
package excep;

public class TGIllegalElseException extends Exception {
```

```
        public TGIllegalElseException(int lineNumber) {
                super("Missing 'if' preceding 'else' at line " + lineNumber);
        }
}
```

8.3.7 excep/TGSymbolNotFoundException.java

```
package excep;

public class TGSymbolNotFoundException extends Exception {

        public TGSymbolNotFoundException(String symbolName) {
                super("Invalid symbol '" + symbolName + "'");
        }

        public TGSymbolNotFoundException(String symbolName, int lineNumber) {
                super("Invalid symbol '" + symbolName + "' on line " + lineNumber);
        }
}
```

8.4 Miscellaneous Scripts

8.4.1 Makefile

Is is our Makefile.
Signed: Andrey Falko

```
reg_test = ${SRCDIR}test/reg-test.pl
excep_dir = ${SRCDIR}excep/
gram_dir = ${SRCDIR}gram/
main_dir = ${SRCDIR}main/
misc_dir = ${SRCDIR}misc/
lib_dir = ${SRCDIR}lib/
symtb_dir = ${SRCDIR}symtb/

# Compile Section Below
# Invoke with "make compile"
compile: grammar inter

grammar:
        java org.antlr.Tool $(gram_dir)TableGen.g
        @perl $(misc_dir)buildprep.pl $(gram_dir)TableGenLexer.java $(gram_dir)TableGenParser.java
        javac $(gram_dir)TableGenLexer.java
        javac $(gram_dir)TableGenParser.java

inter:
```

```
	javac $(excep_dir)*.java
	javac $(symtb_dir)*.java
	javac $(main_dir)*.java

clean:
	rm -f $(gram_dir){*.class,*.tokens,*java,TableGen__.g}
	rm -f {$(main_dir),$(symtb_dir),$(excep_dir)}*.class

clean-test:
	rm -f ${SRCDIR}testfail.log

# Testing Section Below
# Invoke with "make test"
test: clean-test eq for-if hello_world lists pascal putinto average for-each func-return njnets
	@echo Finished running all tests.

eq:
	@echo "Running eq test..."
	@${reg_test} eq.tg
	@echo ======================================

for-if:
	@echo "Running for-if test..."
	@${reg_test} for-if.tg
	@echo ======================================

hello_world:
	@echo "Running hello_world test..."
	@${reg_test} hello_world.tg
	@echo ======================================

lists:
	@echo "Running lists test..."
	@${reg_test} lists.tg
	@echo ======================================

pascal:
	@echo "Running pascal test..."
	@${reg_test} pascal.tg
	@echo ======================================

putinto:
	@echo "Running putinto test..."
	@${reg_test} putinto.tg
	@echo ======================================

average:
	@echo "Running average test..."
	@${reg_test} average.tg
	@echo ======================================

for-each:
	@echo "Running for-each test..."
	@${reg_test} for-each.tg
```

```
	@echo ====================================

func-return:
	@echo "Running func-return test..."
	@${reg_test} func-return.tg
	@echo ====================================

njnets:
	@echo "Running njnets test..."
	@${reg_test} njnets.tg
	@echo ====================================
```

8.4.2 misc/buildprep.pl

This modifies the generated Antlr files to work with our code.
Signed: Andrey Falko

```
#!/usr/bin/perl
#
# Make sure that all java files
# are a part of package gram.
#
use strict;

for my $file (@ARGV) {
	open CODE, $file;
	my @code = <CODE>;

	my $sane = 0;
	for (@code) {
		if (/^package gram\;\n$/) {
			$sane = 1;
			last;
		}
	}

	if (! $sane) {
		open OUT, ">$file";
		print OUT "package gram;\n\n";
		for (@code) {
			print OUT $_;
		}
	}

}
```

8.4.3 misc/rembrak.pl

This removes brackets from files that have them.
Signed: Andrey Falko

```
#!/usr/bin/perl

# After getting the pre-processor to work, we needed to
# remove braces and brakets from our examples.
use strict;

my $file = pop @ARGV;

open FILE, $file or die "Unable to open $file: $!\n";

my @FILE = <FILE>;

close FILE;

open OUT, ">$file" or die "Unable to reopen $file: $!\n";
select OUT;

for (@FILE) {
        s/{|}|\[|\]//g;
        print $_;
}
```

8.4.4 misc/tabbify.pl

This replaces spaces with tabs in indentation.
Signed: Andrey Falko

```
#!/usr/bin/perl

# When wrote our examples, we were not careful about the
# fact that our language counts the number of tabs instead
# of spaces. This scipt removes all leading spaces and
# replacs with tabs.
use strict;

my $file = pop @ARGV;

open FILE, $file or die "Unable to open $file: $!\n";

my @FILE = <FILE>;

close FILE;
```

```
open OUT, ">$file" or die "Unable to reopen $file: $!\n";
select OUT;

my $firstIndent = 0;                                                    20
for (@FILE) {
        if ($firstIndent && /^\s+\w/) {
                my $spaces = " " x $firstIndent;
                s/\t/$spaces/g;
        }
        if (/(^\ +)/) {
                my $numSpaces = int (split "", $1);
                $firstIndent = $numSpaces if (!$firstIndent);
                s/^\ +//;
                print "\t" x int ($numSpaces / $firstIndent);           30
                print "$_";
                next;
        }
        print $_;
}
```

8.4.5 test/add-prog.pl

Automatically adds tests to Makefile.
Signed: Andrey Falko

8.5 Testing Scripts

```
#!/usr/bin/perl
#
# This script does one of two things. If passed an agrument is will add the program to the
# test suite.
#
# If no argument is passed, this script will search the samples directory for files that
# part of the testing suite.
#
# RUN THIS FROM TRUNK
                                                                        10
use strict;
use vars qw [ @prog @MAKE @existing ];

@prog = @ARGV if (@ARGV);

if (!@prog) {
        @prog = glob "samples/*.tg";
}
```

```
for (@prog) {
        s/samples\///;
        s/\.tg//;
}

open MAKEFILE, "Makefile" or die "Couldn't open Makefile: $!";
my @MAKE = <MAKEFILE>;
close MAKEFILE;

@existing = &findExistingTests;

for my $prog (@prog) {
        &addProg ($prog) if (! &progExists ($prog));
}


sub findExistingTests {
        my @existing;
        my $test = 0;
        for (@MAKE) {
                $test = 1 if (/^test: /);
                next if (!$test);

                if (/^(.+):/) {
                        push @existing, $1;
                }
        }

        return @existing;
}

sub progExists {
        my $prog = shift;
        for (@existing) {
                return 1 if ($prog eq $_);
        }

        return 0;
}

sub addProg {
        my $prog = shift;

        for (@MAKE) {
                s/^test: (.+)$/test: $1 $prog/;
        }

        push @MAKE, "$prog: \n";
        push @MAKE, "\t\@echo \"Running $prog test. . .\"\n";
        push @MAKE, "\t\@\${reg_test} $prog.tg\n";
        push @MAKE, "\t\@echo ========================================\n";
        push @MAKE, "\n";

        open MAKEFILE, ">Makefile";
```

```
        for (@MAKE) {
                print MAKEFILE $_;
        }
}
```

8.5.1 test/reg-test.pl

Our main regression testing script.
Signed: Andrey Falko

```
#!/usr/bin/perl -w
#
# Regression testing script.
# Takes as input sample program.
# Assumptions:
#       1) Desired output file starts with sample filename and ends with .out.
#       2) Program executed from main project directory (trunk/).
#

use strict;
#use Term::ScreenColor;
use vars qw [ $samp_dir $samp $tableGen $failog ];

$failog = "testfail.log";

$samp_dir = "samples/";
$samp = $samp_dir . $ARGV[0];

$tableGen = "./TableGen";

&testPreprocessor;
&testIntermediate;
&testPlainOut;
&testLatexOut;
&testHtmlOut;

system ("rm -rf $samp_dir*.real");

sub testPreprocessor {
        my $opt = "--only-preprocess";
        my $real_out = `$tableGen $opt $samp`;

        my @real_out = &printReal ($real_out);
        &diff ("$samp.real", "$samp.out.pre", "Preprocessor Output", $real_out, $opt);
}

sub testIntermediate {
        my $opt = "--only-intermediate";
        my $real_out = `$tableGen $opt $samp`;
```

```
        &printReal ($realℓ_out);
        &diff ("$samp.real", "$samp.out.inter", "Intermediate Output", $realℓ_out, $opt);
}

sub testPlainOut {
        my $opt = "";
        my $realℓ_out = `$tablℓeGen $samp`;

        &printReal ($realℓ_out);
        &diff ("$samp.real", "$samp.out", "Plain Text Output", $realℓ_out, $opt);
}

sub testLatexOut {
        my $opt = "--output-type latex";
        my $realℓ_out = `$tablℓeGen $opt $samp`;

        &printReal ($realℓ_out);
        &diff ("$samp.real", "$samp.out.latex", "LaTeX Output", $realℓ_out, $opt);
}

sub testHtmlOut {
        my $opt = "--output-type html";
        my $realℓ_out = `$tablℓeGen $opt $samp`;

        &printReal ($realℓ_out);
        &diff ("$samp.real", "$samp.out.html", "HTML Output", $realℓ_out, $opt);
}

# Helper routines.
sub printReal {
        my $out = shift;

        open REAL, ">$samp.real" or die "Can't open $samp.real: $!";

        print REAL $out;
}

sub diff {
        my $realℓ = shift;
        my $good = shift;
        my $type = shift;
        my $out = shift;
        my $opt = shift;

        if (! -e $good) {
                print "Nothing to compare: need $good: ";
                print "FAIL\n";
                return;
        }

        my $diff = `diff $realℓ $good`;
        my @diff = split /\n/, $diff;

        #my $scr = new Term::ScreenColor;
```

```
        if (!@diff) {
                # Real output matched desired output.
                print "$type Test: ";
                #$scr->raw ();
                #$scr->putcolored('green', "OK\n");
                print "OK\n";                                                    100
        } else {
                print "$type Test: FAIL (see $faillog)\n";
                open FAIL, ">>testfail.log" or die "Can't open testfail.log: $!";
                print FAIL "=====$type Test Failed on $samp=====\n\nDiff: \n";
                print FAIL "$_\n" for (@diff);
                print FAIL "\n\nCommand run: $tableGen $opt $samp\n";
                print FAIL "\n\nReal output: \n";
                print FAIL $_ for ($out);
                print FAIL "\n\nExpected output: \n";
                open EXP, "$good";                                               110
                print FAIL <EXP>;
                print FAIL "\n" x 5;
                close FAIL;
                close EXP;
        }
}
```

8.6 Project Log

```
------------------------------------------------------------------------
r257 | andrey | 2007-12-17 14:02:00 -0500 (Mon, 17 Dec 2007) | 1 line

Added hash example.
------------------------------------------------------------------------
r256 | andrey | 2007-12-17 02:33:19 -0500 (Mon, 17 Dec 2007) | 1 line

Reduced Pascal by one line.
------------------------------------------------------------------------
r255 | andrey | 2007-12-17 01:31:31 -0500 (Mon, 17 Dec 2007) | 1 line           10

Removed stale output code. Removed completely stale examples. Added
desired output for all existing tests.
------------------------------------------------------------------------
r254 | andrey | 2007-12-17 00:35:27 -0500 (Mon, 17 Dec 2007) | 1 line

Fixed division's ability to output integers. Fixed bug with function
calls.
------------------------------------------------------------------------
r253 | andrey | 2007-12-16 19:44:41 -0500 (Sun, 16 Dec 2007) | 1 line           20

Fixed indentation.
------------------------------------------------------------------------
r252 | andrey | 2007-12-16 19:43:29 -0500 (Sun, 16 Dec 2007) | 1 line
```

```
Updated tutorial for new example.
------------------------------------------------------------------------
r251 | Del | 2007=12=16 18:10:12 =0500 (Sun, 16 Dec 2007) | 1 line

Credited Andrey with writing most of the final writeup and noted that I
did the LRM rewrite
------------------------------------------------------------------------
r250 | andrey | 2007=12=16 17:30:21 =0500 (Sun, 16 Dec 2007) | 1 line

Updated documenation and syned trunk.
------------------------------------------------------------------------
r249 | 4115 | 2007=12=16 17:27:39 =0500 (Sun, 16 Dec 2007) | 1 line


------------------------------------------------------------------------
r248 | andrey | 2007=12=16 05:10:59 =0500 (Sun, 16 Dec 2007) | 1 line

Finished example.
------------------------------------------------------------------------
r247 | andrey | 2007=12=16 04:14:46 =0500 (Sun, 16 Dec 2007) | 1 line

Synced branch.
------------------------------------------------------------------------
r246 | andrey | 2007=12=16 04:13:12 =0500 (Sun, 16 Dec 2007) | 1 line

Will finish Nets example tommorrow, but storing for it now.
------------------------------------------------------------------------
r245 | andrey | 2007=12=16 04:12:44 =0500 (Sun, 16 Dec 2007) | 1 line

Implemented round function.
------------------------------------------------------------------------
r244 | andrey | 2007=12=16 02:42:00 =0500 (Sun, 16 Dec 2007) | 1 line

Fixed return in doFuncCall. (Re=doing because I synced in wrong branch).
------------------------------------------------------------------------
r243 | andrey | 2007=12=16 02:39:16 =0500 (Sun, 16 Dec 2007) | 1 line

Fixed return in doFuncCall.
------------------------------------------------------------------------
r242 | andrey | 2007=12=15 23:36:11 =0500 (Sat, 15 Dec 2007) | 1 line

Previous commit bad because I missed a paren.
------------------------------------------------------------------------
r241 | andrey | 2007=12=15 23:35:00 =0500 (Sat, 15 Dec 2007) | 1 line

Put debugging statement under debug conditional.
------------------------------------------------------------------------
r240 | andrey | 2007=12=15 23:32:42 =0500 (Sat, 15 Dec 2007) | 1 line

Synced trunk.
------------------------------------------------------------------------
r239 | Del | 2007=12=15 19:10:18 =0500 (Sat, 15 Dec 2007) | 1 line

Fixed Integer/Double arithmetic
```

```
------------------------------------------------------------------------
r238 | Del | 2007=12=15 17:33:20 =0500 (Sat, 15 Dec 2007) | 1 line

Fixes return statements again in the trunk this time . . .
------------------------------------------------------------------------
r237 | Del | 2007=12=15 17:32:03 =0500 (Sat, 15 Dec 2007) | 1 line

Fixed returns . . . again.
------------------------------------------------------------------------
r236 | andrey | 2007=12=15 14:19:20 =0500 (Sat, 15 Dec 2007) | 1 line

Modifications to final report.
------------------------------------------------------------------------
r235 | andrey | 2007=12=15 11:43:44 =0500 (Sat, 15 Dec 2007) | 1 line

Synced branch.
------------------------------------------------------------------------
r234 | andrey | 2007=12=15 11:43:00 =0500 (Sat, 15 Dec 2007) | 1 line

Pascal works\!
------------------------------------------------------------------------
r233 | 4115 | 2007=12=14 23:17:57 =0500 (Fri, 14 Dec 2007) | 1 line


------------------------------------------------------------------------
r232 | 4115 | 2007=12=14 22:17:42 =0500 (Fri, 14 Dec 2007) | 1 line


------------------------------------------------------------------------
r231 | andrey | 2007=12=14 20:56:17 =0500 (Fri, 14 Dec 2007) | 1 line

Syned branch.
------------------------------------------------------------------------
r230 | andrey | 2007=12=14 20:41:29 =0500 (Fri, 14 Dec 2007) | 1 line

Syned trunk. Added example.
------------------------------------------------------------------------
r229 | andrey | 2007=12=14 19:49:47 =0500 (Fri, 14 Dec 2007) | 1 line

Added for=each example.
------------------------------------------------------------------------
r228 | Del | 2007=12=14 18:44:35 =0500 (Fri, 14 Dec 2007) | 1 line

Fixed my broken addition of string comparison
------------------------------------------------------------------------
r227 | Del | 2007=12=14 18:31:34 =0500 (Fri, 14 Dec 2007) | 1 line

Added '!=' to the isOperator method
------------------------------------------------------------------------
r226 | Del | 2007=12=14 18:29:04 =0500 (Fri, 14 Dec 2007) | 1 line

Just had to figure out why foreach loops couldn't have internal scopes
properly . . . just passed the wrong child to buildScopes
========================================================================
```

r225 | 4115 | 2007=12=14 13:58:27 =0500 (Fri, 14 Dec 2007) | 2 lines

_avg, _sum added
_length, _tail fixed
==
r224 | Del | 2007=12=14 12:47:56 =0500 (Fri, 14 Dec 2007) | 1 line

Foreach loops working, debugging ifs
==
r223 | andrey | 2007=12=14 11:44:30 =0500 (Fri, 14 Dec 2007) | 1 line

Fixed mistake in getDayOfWeek.
==
r222 | andrey | 2007=12=14 02:07:13 =0500 (Fri, 14 Dec 2007) | 1 line

Fixed comment.
==
r221 | andrey | 2007=12=14 01:42:17 =0500 (Fri, 14 Dec 2007) | 1 line

Tried to get reserved functions working.
==
r220 | andrey | 2007=12=14 00:11:16 =0500 (Fri, 14 Dec 2007) | 1 line

Added debug statements to symbol table.
==
r219 | andrey | 2007=12=14 00:03:50 =0500 (Fri, 14 Dec 2007) | 1 line

Added debugging flag to main.
==
r218 | Del | 2007=12=13 23:39:28 =0500 (Thu, 13 Dec 2007) | 3 lines

Fixed the bug with **return** statements that clobbered **return** values when
popping off the scope == since we re=use scopes, popScope was zeroing
all the elements, and since the **return** value was passed by reference,
it too was zeroed

BUT NO MORE!
==
r217 | andrey | 2007=12=13 21:14:44 =0500 (Thu, 13 Dec 2007) | 1 line

Syncing trunk.
==
r216 | Del | 2007=12=13 20:52:11 =0500 (Thu, 13 Dec 2007) | 2 lines

Fixed many bugs in the interpreter that only came up when executing
calendar.tg:

==
r215 | Del | 2007=12=13 20:51:40 =0500 (Thu, 13 Dec 2007) | 4 lines

Updated the abstract symbol table to prevent adding of function scopes
as children of the global scope to prevent messing up the
`"getNextChildScope"` on it **for** loops, etc.

Deleted the method isGlobal from TGScope since the test **for** a null
parent was no longer valid.

```
------------------------------------------------------------------------
r214 | andrey | 2007-12-13 20:28:21 -0500 (Thu, 13 Dec 2007) | 1 line

Fixed abstract symtb to allow reserved functions.
------------------------------------------------------------------------
r213 | andrey | 2007-12-13 20:08:28 -0500 (Thu, 13 Dec 2007) | 1 line

Removing my dumb comment.
------------------------------------------------------------------------
r212 | andrey | 2007-12-13 20:00:06 -0500 (Thu, 13 Dec 2007) | 1 line

Skeleton for doing reserved functions.
------------------------------------------------------------------------
r211 | andrey | 2007-12-13 19:50:48 -0500 (Thu, 13 Dec 2007) | 1 line

Syned trunk.
------------------------------------------------------------------------
r210 | Del | 2007-12-13 19:26:07 -0500 (Thu, 13 Dec 2007) | 1 line

Changed a rather obscured bug in the interpreter -- pushing the
function's scope onto the scope stack made applicative order
evaluation impossible if the name of the parameter was the name of the
identifier passed as its value
------------------------------------------------------------------------
r209 | andrey | 2007-12-13 17:56:54 -0500 (Thu, 13 Dec 2007) | 1 line

Fixed calendar.tg
------------------------------------------------------------------------
r208 | Del | 2007-12-13 17:48:20 -0500 (Thu, 13 Dec 2007) | 1 line

Added numerical array references to the grammar
------------------------------------------------------------------------
r207 | andrey | 2007-12-13 17:30:18 -0500 (Thu, 13 Dec 2007) | 1 line

Synced trunk.
------------------------------------------------------------------------
r206 | andrey | 2007-12-13 17:26:48 -0500 (Thu, 13 Dec 2007) | 1 line

Updated documentation.
------------------------------------------------------------------------
r205 | Del | 2007-12-13 17:21:21 -0500 (Thu, 13 Dec 2007) | 1 line

Lexer/Parser files corresponding to Andre's addition of the != sign to
the grammar
------------------------------------------------------------------------
r204 | andrey | 2007-12-13 17:18:49 -0500 (Thu, 13 Dec 2007) | 1 line

Removing stale directory.
------------------------------------------------------------------------
r203 | Del | 2007-12-13 17:15:01 -0500 (Thu, 13 Dec 2007) | 1 line
```

All these files edited in some way to implement hash assignment and
the '.' operator **for** referencing elements in a list by their key
==
r202 | andrey | 2007=12=13 17:08:29 =0500 (Thu, 13 Dec 2007) | 1 line

A few edit to LRM.
==
r201 | andrey | 2007=12=13 16:59:28 =0500 (Thu, 13 Dec 2007) | 1 line

Updated final report.
==
r200 | andrey | 2007=12=13 16:38:06 =0500 (Thu, 13 Dec 2007) | 1 line

Grammar lost != :'(
==
r199 | andrey | 2007=12=13 16:25:58 =0500 (Thu, 13 Dec 2007) | 1 line

Removing control.tg; it was initally designed to fail.
==
r198 | andrey | 2007=12=13 16:21:15 =0500 (Thu, 13 Dec 2007) | 1 line

Sync trunk and branch.
==
r197 | Del | 2007=12=13 15:10:39 =0500 (Thu, 13 Dec 2007) | 3 lines

Updated the grammar file to build the tree for list key specification
appropriately and updated the lexer/parser files corresponding to the
grammar update.

I also modified an exception call in TGAbstractSymbolTable and edited
its comments as well as edited Interpreter to allow comparison of
strings as specified in the LRM
==
r196 | andrey | 2007=12=13 15:04:26 =0500 (Thu, 13 Dec 2007) | 1 line

Fixed samples and desired output.
==
r195 | andrey | 2007=12=13 14:49:17 =0500 (Thu, 13 Dec 2007) | 1 line

Removed dependency on external module.
==
r194 | andrey | 2007=12=13 14:48:54 =0500 (Thu, 13 Dec 2007) | 1 line

Removed dependency on external module.
==
r193 | Del | 2007=12=13 14:44:49 =0500 (Thu, 13 Dec 2007) | 3 lines

Did a quick re=write of sec. 5.2 == separated by person and clarified
symbol table vs. interpreter language

also committing the new PDF
==
r192 | andrey | 2007=12=13 13:25:27 =0500 (Thu, 13 Dec 2007) | 1 line

```
Minor edit to final report.
------------------------------------------------------------------------
r191 | andrey | 2007=12=13 03:22:10 =0500 (Thu, 13 Dec 2007) | 1 line

Hello world passes all the way through.                                  300
------------------------------------------------------------------------
r190 | andrey | 2007=12=13 03:21:41 =0500 (Thu, 13 Dec 2007) | 1 line

Fixed two more bug with test script.
------------------------------------------------------------------------
r189 | andrey | 2007=12=13 03:04:37 =0500 (Thu, 13 Dec 2007) | 1 line

Fixed bug in testing script.
------------------------------------------------------------------------
r188 | andrey | 2007=12=13 02:21:54 =0500 (Thu, 13 Dec 2007) | 1 line   310

Fixed up testing script. Created desired preprocessor output. Automated
addition of tests to Makefile.
------------------------------------------------------------------------
r187 | andrey | 2007=12=13 01:02:21 =0500 (Thu, 13 Dec 2007) | 1 line

Draft of first 5 chapters of final report.
------------------------------------------------------------------------
r186 | Del | 2007=12=12 22:39:52 =0500 (Wed, 12 Dec 2007) | 1 line
                                                                         320
Got function calls working == still need to expand the put=ops to put
into _next_row/_next_col and also need to implement foreach loops
------------------------------------------------------------------------
r185 | Del | 2007=12=12 21:23:27 =0500 (Wed, 12 Dec 2007) | 1 line

Working if/elseif/else as well as some loop debugging
------------------------------------------------------------------------
r184 | Del | 2007=12=12 21:02:30 =0500 (Wed, 12 Dec 2007) | 1 line

Implemented conditionals and debugged loops == added a child scope       330
iterator reset method to TGScope for use when interpreting loops
------------------------------------------------------------------------
r183 | andrey | 2007=12=12 20:44:37 =0500 (Wed, 12 Dec 2007) | 1 line

Fixed lines bigger than 125 chars.
------------------------------------------------------------------------
r182 | Del | 2007=12=12 16:35:04 =0500 (Wed, 12 Dec 2007) | 1 line

Debugged for loops == I needed to make two separate interpretBlock and
interpretStatement methods because interpret was only executing          340
CHILDREN of the beginning and ending expressions in the for loop
declaration . . . not good, but fixed =)
------------------------------------------------------------------------
r181 | andrey | 2007=12=12 13:36:21 =0500 (Wed, 12 Dec 2007) | 1 line

Block Diagram added.
------------------------------------------------------------------------
r180 | andrey | 2007=12=12 12:26:38 =0500 (Wed, 12 Dec 2007) | 1 line
```

```
Outline completed.
========================================================================
r179 | Del | 2007=12=12 06:05:18 =0500 (Wed, 12 Dec 2007) | 1 line

Huge changes to the final report tex file == also added the generated
PDF.
========================================================================
r178 | andrey | 2007=12=12 04:19:50 =0500 (Wed, 12 Dec 2007) | 1 line

Syncing branch.
========================================================================
r177 | andrey | 2007=12=12 04:18:30 =0500 (Wed, 12 Dec 2007) | 1 line

Fixed TableGen for !=, if statements, new example.
========================================================================
r176 | andrey | 2007=12=12 00:32:52 =0500 (Wed, 12 Dec 2007) | 1 line

Synced trunk and branches.
========================================================================
r175 | 4115 | 2007=12=12 00:01:37 =0500 (Wed, 12 Dec 2007) | 1 line


========================================================================
r174 | andrey | 2007=12=11 18:56:17 =0500 (Tue, 11 Dec 2007) | 1 line

Small adjustments to Makefile and regression testing script.
========================================================================
r173 | andrey | 2007=12=11 18:38:43 =0500 (Tue, 11 Dec 2007) | 1 line

Regresssion script now tests three stages and three output types.
========================================================================
r172 | andrey | 2007=12=11 15:48:44 =0500 (Tue, 11 Dec 2007) | 1 line

Synced trunk.
========================================================================
r171 | andrey | 2007=12=11 15:48:04 =0500 (Tue, 11 Dec 2007) | 1 line

Fixed output of lists.
========================================================================
r170 | Del | 2007=12=11 15:45:43 =0500 (Tue, 11 Dec 2007) | 1 line


Fixed an issue with non=typechecked parameters in function definition
processing == they were not added to the function's scope
========================================================================
r169 | andrey | 2007=12=11 15:18:56 =0500 (Tue, 11 Dec 2007) | 1 line

Syned trunk
========================================================================
r168 | Del | 2007=12=11 15:10:07 =0500 (Tue, 11 Dec 2007) | 3 lines

Many updates, mostly to the interpreter. I did some interpreter
debugging and found many bugs resulting from checking "type" instead
of "runtime type" in a few situations and other things.
```

I have also changed the grammar file to **include** LOGICAL_OPs as well
as BOOLEAN_OPs in expressions
==
r167 | andrey | 2007=12=11 02:28:52 =0500 (Tue, 11 Dec 2007) | 1 line

Made vector work. Fixed up print statements. Makefile changes.
==
r166 | andrey | 2007=12=10 23:37:04 =0500 (Mon, 10 Dec 2007) | 1 line

Synced trunk and played with hello world.
==
r165 | Del | 2007=12=10 22:59:09 =0500 (Mon, 10 Dec 2007) | 3 lines

Generally have Hello World working with the exception of actually
putting the value into the table == see the method below doPutOp to
test

Also fixed some type issues with casting in the toString methods of
things and changed exception handling so exceptions generally don't
inherit from RuntimeException anymore
==
r164 | andrey | 2007=12=10 21:41:27 =0500 (Mon, 10 Dec 2007) | 1 line

Adding final report.
==
r163 | andrey | 2007=12=10 21:40:06 =0500 (Mon, 10 Dec 2007) | 1 line

Syncing branch with new examples.
==
r162 | andrey | 2007=12=10 21:38:28 =0500 (Mon, 10 Dec 2007) | 1 line

Fixed samples for slight grammar change.
==
r161 | andrey | 2007=12=10 21:30:35 =0500 (Mon, 10 Dec 2007) | 1 line

Updated preproccessor and synced branch and trunk.
==
r160 | Del | 2007=12=10 20:51:18 =0500 (Mon, 10 Dec 2007) | 1 line

Updated grammar files to reflect proper precedence and implementation
of ?all? operators
==
r159 | Del | 2007=12=10 20:46:49 =0500 (Mon, 10 Dec 2007) | 1 line

I messed up my code base == committing what I have and refreshing it
==
r158 | andrey | 2007=12=10 19:27:46 =0500 (Mon, 10 Dec 2007) | 1 line

Added exception files that were not under revision control.
==
r157 | andrey | 2007=12=10 19:26:51 =0500 (Mon, 10 Dec 2007) | 1 line

Forgot to uncomment debugging code.
==

```
r156 | andrey | 2007=12=10 19:25:47 =0500 (Mon, 10 Dec 2007) | 1 line

Synced trunk.
------------------------------------------------------------------------
r155 | andrey | 2007=12=10 19:24:19 =0500 (Mon, 10 Dec 2007) | 1 line

Html output completed.
------------------------------------------------------------------------
r154 | Del | 2007=12=10 18:55:14 =0500 (Mon, 10 Dec 2007) | 7 lines

Updated TGAbstractSymbolTable to do more static=semantic analysis
checking of function calls (with respect to parameter counts, etc.)

Updated Interpreter.java == filled in a lot of utility stuff to work
up to assignment including function calls that are the final part
needed for evaluating expressions (I believe)

In the process I added some functionality to TGHash == just a
static method that returns a TGHash given a string representation of
the value and a name for the hash object.

See e=mails for more info.
------------------------------------------------------------------------
r153 | Del | 2007=12=10 02:00:30 =0500 (Mon, 10 Dec 2007) | 3 lines

in Interpreter.java, started filling in the doFuncDef so assignment
can be fully implemented, added a utility function for only setting a
symbol value within the most recent scope.

in TGAbstractSymbolTable.java added a check for function parameters so
that they are either all named or all unnamed
------------------------------------------------------------------------
r152 | andrey | 2007=12=09 21:11:35 =0500 (Sun, 09 Dec 2007) | 1 line

Sync branch.
------------------------------------------------------------------------
r151 | 4115 | 2007=12=09 20:27:34 =0500 (Sun, 09 Dec 2007) | 1 line


------------------------------------------------------------------------
r150 | Del | 2007=12=09 13:11:58 =0500 (Sun, 09 Dec 2007) | 2 lines

wrote subtract

------------------------------------------------------------------------
r149 | Del | 2007=12=09 13:02:56 =0500 (Sun, 09 Dec 2007) | 1 line

Added a sample addition function for use in the expression evaluation
routine, also added some functionality/constructors to TGHash and TGNum
------------------------------------------------------------------------
r148 | andrey | 2007=12=09 12:53:39 =0500 (Sun, 09 Dec 2007) | 1 line

Synced more changes.
------------------------------------------------------------------------
```

```
r147 | andrey | 2007=12=09 12:52:46 =0500 (Sun, 09 Dec 2007) | 1 line

Syned branch and updated branch's samples files.
========================================================================
r146 | Del | 2007=12=09 12:39:12 =0500 (Sun, 09 Dec 2007) | 1 line

Added stuff to help the interpreter, including methods for manipulating
a runtime type rather than just the type restriction
========================================================================
r145 | Del | 2007=12=09 09:21:39 =0500 (Sun, 09 Dec 2007) | 1 line

forgot that instanceof only looks for valid upcasts . . .
========================================================================
r144 | Del | 2007=12=09 09:14:05 =0500 (Sun, 09 Dec 2007) | 1 line

Updated the interpreter to have a lone method for setting values in
the symbol table since it turns out that the way for doing it is now
through a uniform interface with TGHash == and hence it's update as
well
========================================================================
r143 | andrey | 2007=12=09 00:51:32 =0500 (Sun, 09 Dec 2007) | 1 line

Fixed up samples and synced eclipse branch.
========================================================================
r142 | Del | 2007=12=08 19:52:01 =0500 (Sat, 08 Dec 2007) | 1 line

Added more methods and started filling in the interpreter
========================================================================
r141 | andrey | 2007=12=08 18:19:36 =0500 (Sat, 08 Dec 2007) | 1 line

Sync branch with trunk and vice versa.
========================================================================
r140 | andrey | 2007=12=08 18:16:00 =0500 (Sat, 08 Dec 2007) | 1 line

Created re=sync.sh for backwards sync.
========================================================================
r139 | Del | 2007=12=08 18:09:32 =0500 (Sat, 08 Dec 2007) | 3 lines

Made everything talk in the language of hashes instead of symbols ==
now every variable is a hash

There is also a new exception == TGTypeException == you can call it
with one constructor to have only a message, or with the other to also
give a number line. It extends RuntimeException so it does not
necessarily have to be caught.
========================================================================
r138 | andrey | 2007=12=08 17:39:29 =0500 (Sat, 08 Dec 2007) | 1 line

Minor comment.
========================================================================
r137 | andrey | 2007=12=08 17:36:14 =0500 (Sat, 08 Dec 2007) | 1 line

Preprocessor complete and debugged; minor fix to pascal example.
========================================================================
```

```
r136 | andrey | 2007=12=08 15:23:37 =0500 (Sat, 08 Dec 2007) | 1 line

Created tabbify scipt and fixed pascal.tg.
========================================================================
r135 | andrey | 2007=12=08 14:51:49 =0500 (Sat, 08 Dec 2007) | 1 line

Created sync script, and synced trunk to latest EclipseProject branch.
========================================================================
r134 | andrey | 2007=12=08 14:31:51 =0500 (Sat, 08 Dec 2007) | 1 line

Moving deleted files.
========================================================================
r133 | Del | 2007=12=08 11:34:19 =0500 (Sat, 08 Dec 2007) | 1 line

added file I missed
========================================================================
r132 | Del | 2007=12=08 11:33:29 =0500 (Sat, 08 Dec 2007) | 1 line

Attempting to make this an easily=downloadable eclipse project
========================================================================
r131 | Del | 2007=12=08 11:25:09 =0500 (Sat, 08 Dec 2007) | 1 line


========================================================================
r130 | andrey | 2007=12=08 10:49:39 =0500 (Sat, 08 Dec 2007) | 1 line

Fixed up Interpretor and Makefile.
========================================================================
r129 | andrey | 2007=12=08 10:37:26 =0500 (Sat, 08 Dec 2007) | 1 line

Moved work from branch to main.
========================================================================
r128 | Del | 2007=12=08 10:23:33 =0500 (Sat, 08 Dec 2007) | 1 line

Fixed a comment
========================================================================
r127 | 4115 | 2007=12=08 04:17:07 =0500 (Sat, 08 Dec 2007) | 1 line


========================================================================
r126 | Del | 2007=12=08 03:56:41 =0500 (Sat, 08 Dec 2007) | 3 lines

So = the abstract symbol table building is almost completely done.
There are substantial updates to the grammar file, and the
corresponding lexer and parser files are posted with it. Many things
are checked, and the role of the tree validation method has changed
slightly.

The main buildScope method has also changed to allow the caller to
make the scope rather than automatically make it and add stuff to a
current scope == this is needed because of adding symbols to a
scope before building it up such as function parameters and variable
declarations in for loops, etc.
========================================================================
```

```
r125 | andrey | 2007=12=08 03:13:12 =0500 (Sat, 08 Dec 2007) | 1 line

Nested old trunk directory. . .not supposed to be here.
------------------------------------------------------------------------
r124 | andrey | 2007=12=08 03:07:52 =0500 (Sat, 08 Dec 2007) | 1 line

Removing files that are not supposed to be under version control.
------------------------------------------------------------------------
r123 | andrey | 2007=12=08 03:03:10 =0500 (Sat, 08 Dec 2007) | 1 line

Latex chart implemented.
------------------------------------------------------------------------
r122 | Del | 2007=12=07 21:48:01 =0500 (Fri, 07 Dec 2007) | 1 line

Fixed issues with the precedence of operators == we needed the putop
to be of higher precedence than anything in a simple expression, so it
not is . . . and is, in fact, a statement in and of itself as of now.
------------------------------------------------------------------------
r121 | andrey | 2007=12=05 02:36:23 =0500 (Wed, 05 Dec 2007) | 1 line

Copied code from branch.
------------------------------------------------------------------------
r120 | andrey | 2007=12=04 09:35:40 =0500 (Tue, 04 Dec 2007) | 1 line

Removing Executor.java
------------------------------------------------------------------------
r119 | 4115 | 2007=12=04 02:56:38 =0500 (Tue, 04 Dec 2007) | 1 line

I filled in the outputTable() method in the Interpreter class.
------------------------------------------------------------------------
r118 | Del | 2007=12=03 21:26:37 =0500 (Mon, 03 Dec 2007) | 1 line

added comment for stuff to work on
------------------------------------------------------------------------
r117 | Del | 2007=12=01 15:15:25 =0500 (Sat, 01 Dec 2007) | 1 line


------------------------------------------------------------------------
r116 | andrey | 2007=12=01 14:57:26 =0500 (Sat, 01 Dec 2007) | 1 line

Came back to r105.
------------------------------------------------------------------------
r115 | andrey | 2007=12=01 14:48:26 =0500 (Sat, 01 Dec 2007) | 1 line

Reverting to r105.
------------------------------------------------------------------------
r114 | Del | 2007=11=30 21:41:58 =0500 (Fri, 30 Nov 2007) | 5 lines

Added two exceptions that should be thrown by the interpreter and made
a very meat=y outline for the interpreter.

Grammar changed a tad, but only what text is in the tree (not its
structure), and new lexer/parser files were generated accordingly.
```

```
There has also been a refactoring of the TGAbstractSymbolTable code for
clarity.
------------------------------------------------------------------------
r113 | Del | 2007=11=30 10:46:22 =0500 (Fri, 30 Nov 2007) | 1 line

OK= a new branch of what I did instead of all the crap so the
repository can be rolled back to rev 105
------------------------------------------------------------------------
r112 | Del | 2007=11=30 09:55:40 =0500 (Fri, 30 Nov 2007) | 1 line

Undoing my re=versioning
------------------------------------------------------------------------
r111 | Del | 2007=11=30 02:08:42 =0500 (Fri, 30 Nov 2007) | 1 line

For some reason I am having to commit this again
------------------------------------------------------------------------
r110 | Del | 2007=11=30 02:08:23 =0500 (Fri, 30 Nov 2007) | 1 line

New main testing file
------------------------------------------------------------------------
r109 | Del | 2007=11=30 02:07:50 =0500 (Fri, 30 Nov 2007) | 5 lines

This is a major revision of the "abstract" symbol table code. I've
sent out a good e=mail with even more information.

I have changed the grammar file a bit to make the job a bit easier
when verifying things in the static=semantic analysis, and there
are also corresponding lexer/parser files that are required for
proper compilation.

There is also a new main TableGen.java file for running the new
test through the building of the "abstract" symbol table. There
are nice debugging messages and whatnot.
------------------------------------------------------------------------
r108 | Del | 2007=11=30 02:00:46 =0500 (Fri, 30 Nov 2007) | 1 line


------------------------------------------------------------------------
r107 | andrey | 2007=11=29 17:33:26 =0500 (Thu, 29 Nov 2007) | 1 line

Commented on my code.
------------------------------------------------------------------------
r106 | Del | 2007=11=29 16:52:56 =0500 (Thu, 29 Nov 2007) | 1 line

Replaced pasted public items from gram/TableGenLexer.java with
references to the file itself
------------------------------------------------------------------------
r105 | andrey | 2007=11=27 19:17:51 =0500 (Tue, 27 Nov 2007) | 1 line

Fixed breakage after symbol table update.
------------------------------------------------------------------------
r104 | andrey | 2007=11=27 19:02:57 =0500 (Tue, 27 Nov 2007) | 1 line

Removed unneeded sync.sh
```

```
------------------------------------------------------------------------
r103 | 4115 | 2007=11=26 18:23:53 =0500 (Mon, 26 Nov 2007) | 1 line

------------------------------------------------------------------------
r102 | 4115 | 2007=11=26 18:21:25 =0500 (Mon, 26 Nov 2007) | 1 line


------------------------------------------------------------------------
r101 | andrey | 2007=11=25 13:59:47 =0500 (Sun, 25 Nov 2007) | 1 line

Fixed don't output debugging stuff by default; Hello, world works.
------------------------------------------------------------------------
r100 | andrey | 2007=11=25 05:06:09 =0500 (Sun, 25 Nov 2007) | 1 line

Deleting Deprecated Files.
------------------------------------------------------------------------
r99 | andrey | 2007=11=25 02:04:45 =0500 (Sun, 25 Nov 2007) | 1 line

Some comments and slightly improved debugging.
------------------------------------------------------------------------
r98 | andrey | 2007=11=25 01:51:39 =0500 (Sun, 25 Nov 2007) | 1 line

Adding new test prog.
------------------------------------------------------------------------
r97 | andrey | 2007=11=25 01:49:44 =0500 (Sun, 25 Nov 2007) | 1 line

Interpretor can add, subtract, and put into coordinates.
------------------------------------------------------------------------
r96 | andrey | 2007=11=24 21:24:13 =0500 (Sat, 24 Nov 2007) | 1 line

Find leftmost member.
------------------------------------------------------------------------
r95 | andrey | 2007=11=24 04:56:30 =0500 (Sat, 24 Nov 2007) | 1 line

Wrote framework to generate basic output in plain text.
------------------------------------------------------------------------
r94 | andrey | 2007=11=20 03:00:18 =0500 (Tue, 20 Nov 2007) | 1 line

Reverting accidental commit of local changes.
------------------------------------------------------------------------
r93 | andrey | 2007=11=20 02:59:21 =0500 (Tue, 20 Nov 2007) | 1 line

Updated build environment for Daniel's Symbol table submission.
------------------------------------------------------------------------
r92 | 4115 | 2007=11=20 02:28:19 =0500 (Tue, 20 Nov 2007) | 1 line


------------------------------------------------------------------------
r91 | Del | 2007=11=18 13:45:11 =0500 (Sun, 18 Nov 2007) | 1 line

allows arbitrary expressions inside coordinate specifications
------------------------------------------------------------------------
r90 | andrey | 2007=11=16 19:08:14 =0500 (Fri, 16 Nov 2007) | 1 line
```

Merged **pascal** example.
==
r89 | Del | 2007=11=16 10:01:13 =0500 (Fri, 16 Nov 2007) | 1 line

See the e=mail I sent out on this date
==
r88 | Del | 2007=11=15 19:10:02 =0500 (Thu, 15 Nov 2007) | 1 line

790

==
r87 | andrey | 2007=11=14 20:44:24 =0500 (Wed, 14 Nov 2007) | 1 line

Copied new files to build locations.
==
r86 | Del | 2007=11=14 19:18:00 =0500 (Wed, 14 Nov 2007) | 1 line

added basic functionality to the TGScope and added helpful comments
to the beginning of the build() method in TGSymbolTable == sick/tired 800
so going to bed
==
r85 | Del | 2007=11=14 14:19:20 =0500 (Wed, 14 Nov 2007) | 1 line

added single=line comments, added parenthetical expressions. The
par expressions add an ambiguity that ANTLR resolves in the required
way, but **if** a non=ambiguous solution can be found it would be preferred
==
r84 | andrey | 2007=11=13 23:37:43 =0500 (Tue, 13 Nov 2007) | 1 line

810

Update clean procedure in Makefile.
==
r83 | andrey | 2007=11=13 23:15:21 =0500 (Tue, 13 Nov 2007) | 1 line

A couple of helpful initial debugging lines.
==
r82 | andrey | 2007=11=13 23:09:04 =0500 (Tue, 13 Nov 2007) | 1 line

Bringing symbol table into build process.
==
r81 | andrey | 2007=11=13 21:38:11 =0500 (Tue, 13 Nov 2007) | 1 line 820

Hide output of make commands.
==
r80 | Del | 2007=11=13 21:04:58 =0500 (Tue, 13 Nov 2007) | 1 line

A good start on the symbol table == implement the build() method
==
r79 | andrey | 2007=11=12 02:21:39 =0500 (Mon, 12 Nov 2007) | 1 line

830

Execute with main file.
==
r78 | 4115 | 2007=11=11 23:32:29 =0500 (Sun, 11 Nov 2007) | 1 line

There was a change made in the packages in TableGenLexer.java and

TableGenParser.java.
----
r77 | 4115 | 2007=11=11 23:13:45 =0500 (Sun, 11 Nov 2007) | 1 line

There was a new main file created.
----
r76 | andrey | 2007=11=10 16:18:03 =0500 (Sat, 10 Nov 2007) | 1 line

Fixed up testing script and makefile.
----
r75 | andrey | 2007=11=10 16:03:53 =0500 (Sat, 10 Nov 2007) | 1 line

Adding a bunch of tests.
----
r74 | Del | 2007=11=10 15:43:45 =0500 (Sat, 10 Nov 2007) | 1 line

testing file I used **while** writing the lexer/parser == assumes the
pre=processor has already been run.
----
r73 | Del | 2007=11=10 15:41:17 =0500 (Sat, 10 Nov 2007) | 1 line

Includes /* */ comments == make sure to tell me of any needed
alterations ASAP!
----
r72 | andrey | 2007=11=10 15:22:03 =0500 (Sat, 10 Nov 2007) | 1 line

Skeleton of main executable and additions to frontend.
----
r71 | andrey | 2007=11=09 03:13:28 =0500 (Fri, 09 Nov 2007) | 1 line

Pre=processor works again.
----
r70 | andrey | 2007=11=06 13:00:08 =0500 (Tue, 06 Nov 2007) | 1 line

Fixed up the pre=processor a bit.
----
r69 | andrey | 2007=11=05 02:09:26 =0500 (Mon, 05 Nov 2007) | 1 line

Fixed up build systems. Updated the pre=processor.
----
r68 | Del | 2007=10=28 14:16:11 =0400 (Sun, 28 Oct 2007) | 1 line

Grammar with precedence=driven expressions and AST building == let's
get the rest done!
----
r67 | Del | 2007=10=28 14:15:06 =0400 (Sun, 28 Oct 2007) | 1 line

Updated lexer/parser files for new grammar w/ AST building
----
r66 | Del | 2007=10=24 23:41:27 =0400 (Wed, 24 Oct 2007) | 1 line

Ok= this time it works for a much larger example
----
r65 | Del | 2007=10=24 22:10:52 =0400 (Wed, 24 Oct 2007) | 3 lines

```
IT WORKS!!!!!!!

HOLY SHIT IT WORKS!
------------------------------------------------------------------------
r64 | Del | 2007=10=22 13:26:46 =0400 (Mon, 22 Oct 2007) | 1 line

Made all things that are only part of tokens be declared as
"fragment"s (antlr 2 equivalent is "protected")
------------------------------------------------------------------------
r63 | Del | 2007=10=21 13:33:56 =0400 (Sun, 21 Oct 2007) | 3 lines

First version of the new SRC directory= TableGenLexer/Parser.java are
outputs of running the TableGen.g file through ANTLR.

Check STDERR for error messages, supply a test file as the first
argument to the program
------------------------------------------------------------------------
r62 | Del | 2007=10=21 08:03:14 =0400 (Sun, 21 Oct 2007) | 1 line

re=added the recognition exception handler
------------------------------------------------------------------------
r61 | Del | 2007=10=20 23:05:36 =0400 (Sat, 20 Oct 2007) | 1 line

added the necessary elements for escaped characters in strings as per
the LRM and put the ASTNodeType=CommonTree in . . . I think that makes
my code print out the AST in the altered version of AntlrTest.java
------------------------------------------------------------------------
r60 | Del | 2007=10=20 23:02:54 =0400 (Sat, 20 Oct 2007) | 1 line

altered to print out the AST!!!
------------------------------------------------------------------------
r59 | Del | 2007=10=20 22:12:36 =0400 (Sat, 20 Oct 2007) | 1 line

updated comments, has hash declarations, verified VALID_CHARS
------------------------------------------------------------------------
r58 | andrey | 2007=10=18 17:01:08 =0400 (Thu, 18 Oct 2007) | 1 line

LRM Final Version.
------------------------------------------------------------------------
r57 | andrey | 2007=10=18 16:59:11 =0400 (Thu, 18 Oct 2007) | 1 line

LRM Final Version.
------------------------------------------------------------------------
r56 | Del | 2007=10=18 08:24:46 =0400 (Thu, 18 Oct 2007) | 1 line

I removed all of my embarrassing comments from the grammar file ==
they don't need to know about our problems quite yet if they don't
bite them first =P
------------------------------------------------------------------------
r55 | andrey | 2007=10=18 02:04:49 =0400 (Thu, 18 Oct 2007) | 1 line

Lrm, pre=final draft.
------------------------------------------------------------------------
```

```
r54 | andrey | 2007=10=17 23:48:41 =0400 (Wed, 17 Oct 2007) | 1 line

Daniel's Edits.
========================================================================
```
r53 | Del | 2007=10=17 22:57:25 =0400 (Wed, 17 Oct 2007) | 5 lines

The refactored and more likely correct version of the language as
supplied to ANTLR. It may be incomplete.

There is only one ambiguity, and according to the error message ANTLR
is automatically disabling the proper match == ie the one that we
do not want to match. This is a problem arising from the > being
used in both hash key assignment and boolean comparison.

The lookahead must be at least 2 because of the '=>' (put into) operator.
```
========================================================================
r52 | Del | 2007=10=17 22:11:14 =0400 (Wed, 17 Oct 2007) | 1 line

Added block definition and began scope. If there are better examples
in the C LRM, someone **else** can add it
```
==
r51 | Del | 2007=10=17 22:01:30 =0400 (Wed, 17 Oct 2007) | 11 lines

Fixed a lot of language, added "an"s, etc.

Changed the string definition to be clearer

Added verb tags to > signs to make LaTeX happy

Reworded many things to condense adjacent sentences that were
lucky to even stand on their own as a sentence

deleted the exponentiation operator to make our lives easier

check the diff **for** the rest
```
========================================================================
r50 | andrey | 2007=10=17 17:20:10 =0400 (Wed, 17 Oct 2007) | 1 line

Sqaure Brackets **for** coordinates.
```
==
r49 | andrey | 2007=10=17 17:17:46 =0400 (Wed, 17 Oct 2007) | 1 line

Sqaure Brackets **for** coordinates.
```
========================================================================
r48 | andrey | 2007=10=17 13:15:46 =0400 (Wed, 17 Oct 2007) | 1 line

Some minor formatting and typo fixes.
```
==
r47 | 4115 | 2007=10=17 12:37:34 =0400 (Wed, 17 Oct 2007) | 1 line


```
========================================================================
r46 | 4115 | 2007=10=17 11:28:18 =0400 (Wed, 17 Oct 2007) | 1 line


```

```
------------------------------------------------------------------------
r45 | andrey | 2007-10-16 02:56:20 -0400 (Tue, 16 Oct 2007) | 1 line

Preprocessor now does things other than tabs (i.e. spaces)
------------------------------------------------------------------------
r44 | andrey | 2007-10-16 02:48:51 -0400 (Tue, 16 Oct 2007) | 1 line

Preprocessor
------------------------------------------------------------------------
r43 | andrey | 2007-10-16 02:48:28 -0400 (Tue, 16 Oct 2007) | 1 line

Preprocessor, draft 2...added some "tough" samples.
------------------------------------------------------------------------
r42 | andrey | 2007-10-16 02:32:35 -0400 (Tue, 16 Oct 2007) | 1 line

Preprocessor, draft.
------------------------------------------------------------------------
r41 | andrey | 2007-10-13 18:18:54 -0400 (Sat, 13 Oct 2007) | 1 line

Added main executable. Testing Email Hook.
------------------------------------------------------------------------
r40 | andrey | 2007-10-13 18:03:27 -0400 (Sat, 13 Oct 2007) | 1 line

Another testing of commit email system.
------------------------------------------------------------------------
r39 | andrey | 2007-10-13 18:02:07 -0400 (Sat, 13 Oct 2007) | 1 line

Testing commit email system.
------------------------------------------------------------------------
r38 | Del | 2007-10-13 16:18:47 -0400 (Sat, 13 Oct 2007) | 1 line

Fixed some spelling and re-worded the list stuff
------------------------------------------------------------------------
r37 | Del | 2007-10-13 16:12:59 -0400 (Sat, 13 Oct 2007) | 1 line

I have edited all material up to the section on "expressions" - we
need to straighten out our definition of strings
------------------------------------------------------------------------
r36 | Del | 2007-10-13 14:23:24 -0400 (Sat, 13 Oct 2007) | 1 line

Replaced "float" with "floating" to prevent errors when ANTLR
automatically generates methods == "float" is a Java keyword
------------------------------------------------------------------------
r35 | andrey | 2007-10-13 04:15:57 -0400 (Sat, 13 Oct 2007) | 1 line

LRM Draft #1.
------------------------------------------------------------------------
r34 | andrey | 2007-10-12 22:23:29 -0400 (Fri, 12 Oct 2007) | 1 line

Re-organization of grammar file.
------------------------------------------------------------------------
r33 | andrey | 2007-10-12 20:00:56 -0400 (Fri, 12 Oct 2007) | 1 line
```

Minor changes to grammar.
==
r32 | andrey | 2007=10=12 19:33:52 =0400 (Fri, 12 Oct 2007) | 1 line

Changed where main is located.
==
r31 | andrey | 2007=10=12 19:18:16 =0400 (Fri, 12 Oct 2007) | 1 line

Hello world sample.
==
r30 | andrey | 2007=10=12 19:17:32 =0400 (Fri, 12 Oct 2007) | 1 line

Lots of changes and re=organization.
==
r29 | andrey | 2007=10=12 18:37:39 =0400 (Fri, 12 Oct 2007) | 1 line

Shifting directies around
==
r28 | Del | 2007=10=07 20:21:51 =0400 (Sun, 07 Oct 2007) | 3 lines

Committing an initial get=up=and=running java front=end: pass it one
argument **for** the file to use as input. It currently only gets things
up and running, but is a good start.

Make sure to download and link to ANTLR jars when compiling.
==
r27 | Del | 2007=10=07 20:16:30 =0400 (Sun, 07 Oct 2007) | 1 line

==
r26 | Del | 2007=10=07 20:13:14 =0400 (Sun, 07 Oct 2007) | 1 line

Structured slightly differently. May be inefficient because I had to
turn on "`backtrack = true`" == I hope he doesn't yell at us. It may
work, may not. I will upload a basic front=end interface program
shortly
==
r25 | andrey | 2007=10=07 14:15:59 =0400 (Sun, 07 Oct 2007) | 1 line

Outline of lrm.
==
r24 | Del | 2007=10=04 08:17:28 =0400 (Thu, 04 Oct 2007) | 3 lines

New folder for the grammar = The grammar checks by ANTLR say it is OK.
Whether or not it is really correct is another story.

It is not fully completed, but is in at least an interesting state.
==
r23 | andrey | 2007=09=28 00:12:19 =0400 (Fri, 28 Sep 2007) | 1 line

Regression test script beginings and hypothetical Makefile.
==
r22 | Del | 2007=09=25 08:52:06 =0400 (Tue, 25 Sep 2007) | 9 lines

OK= I'm just changing a few things. In the calculate pay example, we still had a row.pay =, so I changed it to (value) => row.pay

I added "in" as a keyword **for** using with the foreach

Either I missed the part on Strings on pgs 7 and 8 or it was added . . . but we are not using the list syntax **for** concatenating two things . . . that are lists! I'm putting that in.

I also added some to the "units" part == the idea was more that units are attached to variables so that the units get printed at the end after magic conversions so the user doesn't have to worry about not using the conversion factor.

~!~Del

---

r21 | andrey | 2007=09=25 01:32:29 =0400 (Tue, 25 Sep 2007) | 1 line

Final proposal draft.

---

r20 | Del | 2007=09=23 19:37:58 =0400 (Sun, 23 Sep 2007) | 3 lines

Added notes about why we label parameters in function calls and noted that the "units" section only applied **if** we decided to implement it

~!~Del

---

r19 | andrey | 2007=09=22 19:45:31 =0400 (Sat, 22 Sep 2007) | 1 line

Second Draft.

---

r18 | Del | 2007=09=22 10:37:55 =0400 (Sat, 22 Sep 2007) | 7 lines

Added itemized stuff, fixed spelling and general inconsistencies.

Commented on a super=f'd up code example that we HAVE to look at.

We should seriously consider re=doing ALL code samples except for mine because it is cleanest and closest to what we want =P == or the others should be re=written

~!~Del

---

r17 | andrey | 2007=09=20 00:59:56 =0400 (Thu, 20 Sep 2007) | 1 line

Added unit feature. Ran spell check.

---

r16 | 4115 | 2007=09=18 20:03:37 =0400 (Tue, 18 Sep 2007) | 1 line

Testing Check In.

---

r15 | andrey | 2007=09=18 10:00:36 =0400 (Tue, 18 Sep 2007) | 1 line

Fixed some whitespace issues.

```
========================================================================
r14 | 4115 | 2007=09=17 13:31:35 =0400 (Mon, 17 Sep 2007) | 3 lines

made real lists of items using the "itemize" environment == check out
the "not so short introduction to latex2e" or my e=mail for syntax info

NOTE: LOCALNESS IS FOOLISH AND UNNECESSARY BECAUSE IT IS IMPLIED!
========================================================================
r13 | andrey | 2007=09=16 20:00:59 =0400 (Sun, 16 Sep 2007) | 1 line

Del's Text Integration End.
========================================================================
r12 | andrey | 2007=09=16 18:44:25 =0400 (Sun, 16 Sep 2007) | 1 line

Del's Text Integration Start.
========================================================================
r11 | andrey | 2007=09=16 18:22:18 =0400 (Sun, 16 Sep 2007) | 1 line

Latex Layout.
========================================================================
r10 | andrey | 2007=09=16 18:03:20 =0400 (Sun, 16 Sep 2007) | 1 line

Proposal LaTeXing start.
========================================================================
r9 | 4115 | 2007=09=16 15:51:13 =0400 (Sun, 16 Sep 2007) | 1 line

spelling corrections
========================================================================
r8 | andrey | 2007=09=16 14:01:17 =0400 (Sun, 16 Sep 2007) | 1 line

Variable scoping and local reserved word.
========================================================================
r7 | andrey | 2007=09=16 13:55:39 =0400 (Sun, 16 Sep 2007) | 1 line

Fixed some typos.
========================================================================
r6 | 4115 | 2007=09=15 09:08:07 =0400 (Sat, 15 Sep 2007) | 3 lines

took out commas after stuff already separated by whitespace, added
more structure to specification of detailed language specification
including a section on reserved words.

Delayed putting in stuff on units because we may just have enough on
our plate already
========================================================================
r5 | andrey | 2007=09=14 02:00:07 =0400 (Fri, 14 Sep 2007) | 1 line

1st Draft of Proposal.
========================================================================
r4 | andrey | 2007=09=13 18:31:29 =0400 (Thu, 13 Sep 2007) | 1 line

Start of proposal.
========================================================================
r3 | andrey | 2007=09=13 17:02:36 =0400 (Thu, 13 Sep 2007) | 1 line
```

```
Trunk not truck\!
------------------------------------------------------------------------
r2 | andrey | 2007=09=06 12:54:22 =0400 (Thu, 06 Sep 2007) | 1 line

Testing Check=in.
------------------------------------------------------------------------
r1 | andrey | 2007=09=06 12:52:12 =0400 (Thu, 06 Sep 2007) | 1 line

Initial Import.
------------------------------------------------------------------------
```

1220