gscc
A General Search and Compare Compiler

FINAL REPORT

gscc is a text manipulation language that rivals existing programmatic solutions. It is compact, intuitive and lightweight, giving programmers a means to quickly manipulate their text-based targets.

Eric [G]arrido
Russel [S]antillanes
Casey [C]allendrello
Ho Yin [C]heng

TABLE OF CONTENTS

1 Introduction: The Proposal ..... - 6 -

1.1 An Introduction to gscc ..... - 6 -

1.2 Background ..... - 6 -

1.3 Goals of gscc ..... - 6 -

1.3.1 Simplicity ..... - 6 -

1.3.2 Streamlined Operation ..... - 6 -

1.3.3 Portability ..... - 7 -

1.4 Language Features ..... - 7 -

1.4.1 Small, powerful command set ..... - 7 -

1.4.2 User-defined subroutines ..... - 7 -

1.4.3 Unicode Support ..... - 7 -

1.5 gscc in Practice ..... - 7 -

1.5.1 A First Syntax Example ..... - 7 -

1.5.2 Further Examples ..... - 7 -

2 Language Tutorial ..... - 9 -

2.1 Compiling/Running gscc ..... - 9 -

2.2 Creating a gscc program ..... - 9 -

2.3 Another Example: File Searching ..... - 10 -

2.4 Search and Replacement Program ..... - 11 -

2.5 A more complicated useful example (Apache log parsing) ..... - 13 -

2.6 A really cool example ..... - 14 -

3 Language Reference Manual ..... - 17 -

3.1 Introduction ..... - 17 -

3.2 Lexical Conventions ..... - 17 -

3.2.1 Tokens ..... - 17 -

3.2.2 *Comments* ..... - 17 -

3.2.3 *Identifiers* ..... - 17 -

3.2.4 *Keywords* ..... - 17 -

3.2.5 *Constants* ..... - 18 -

3.2.5.1 Integer Constants ..... - 18 -

3.2.6 *Escape Characters* ..... - 18 -

3.2.7 White space ..... - 18 -

3.2.8 *String Literals* ..... - 18 -

3.2.9 *NULL Literal* ..... - 19 -

3.2.10 *Line Terminator* ..... - 19 -

3.2.11 *Separators* ..... - 19 -

3.3 Error Handling ..... - 19 -

3.4 Identifiers ..... - 19 -

3.4.1 Basic Variables ..... - 20 -

3.4.2 *System Variables* ..... - 21 -

3.5 Expressions ..... - 21 -

3.5.1 *Definition* ..... - 21 -

3.5.2 *Comma Separator* - 21 -
3.5.3 *Evaluation Order* - 22 -
3.5.4 Precedence - 22 -
3.5.5 Unary Minus Operator - 22 -
3.5.6 Logical Not Operator - 22 -
3.5.7 Multiplicative Operators - 22 -
3.5.8 Additive Operators - 23 -
3.5.9 Relational Operators - 23 -
3.5.10 Equality Operators - 24 -
3.5.11 Logical Operators - 24 -
3.6 Syntax Notation - 24 -
3.7 Regular Expression Blocks - 25 -
3.7.1 regexp Blocks - 25 -
3.7.2 Regular Expression - 26 -
3.7.3 Most Commonly used Regular Expressions - 26 -
3.8 Statements - 27 -
3.8.1 Selection Statements - 27 -
3.8.2 Iteration Statements - 28 -
3.8.3 Return Statement - 29 -
3.9 System Commands - 29 -
3.9.1 Operational System Commands - 30 -
3.9.2 Printing System Commands - 32 -
3.10 Functions - 33 -
3.11 Built in functions - 34 -
3.12 Scope - 35 -
4 Project Plan - 36 -
4.1 Team Responsibilities - 36 -
4.2 Programming Style Guide - 36 -
4.2.1 ANTLR coding style: - 36 -
4.2.2 Java coding style: - 36 -
4.2.2.1 Spacing issues: - 36 -
4.2.2.2 Nomenclature: - 36 -
4.3 Project Timeline - 37 -
4.4 Software Project Environment - 37 -
4.4.1 Operating Systems - 37 -
4.4.2 Java 5.0 - 38 -
4.4.3 ANTLR - 38 -
4.4.4 Subversion (SVN) - 38 -
4.4.5 Eclipse - 38 -
4.4.6 Perl - 38 -
4.5 Project Log - 39 -
5 Architectural Design - 40 -
5.1 Block Diagram - 40 -
5.2 Component Interfaces - 40 -
6 Test Plan - 42 -

6.1 Test Suites ..... - 42 -
6.2 Reasoning ..... - 42 -
6.3 Automation ..... - 42 -
6.4 Accountability ..... - 43 -
6.5 Examples ..... - 43 -
6.5.1 Input file -- Mary.txt ..... - 43 -
6.5.2 Test 1 -- if_test1.gscc ..... - 43 -
6.5.3 Test 2 -- if_test2.gscc ..... - 44 -
7 Lessons Learned ..... - 50 -
8 Complete Listing ..... - 52 -
8.1 Input Files ..... - 52 -
8.1.1 Mary.txt ..... - 52 -
8.1.2 helloworld.bf ..... - 52 -
8.2 Script Files ..... - 52 -
8.2.1 compile_gscc ..... - 52 -
8.2.2 run_gscc ..... - 52 -
8.2.3 clean_gscc ..... - 52 -
8.3 Example Files ..... - 53 -
8.3.1 simple.gscc ..... - 53 -
8.3.2 test.gscc ..... - 53 -
8.3.3 brainf_ck.gscc ..... - 55 -
8.4 Main program ..... - 56 -
8.4.1 gscc.java ..... - 56 -
8.4.2 JavaTest.java ..... - 60 -
8.4.3 ParseTester.java ..... - 60 -
8.5 gsccAntlr ..... - 61 -
8.5.1 ccgsGrammar.g ..... - 61 -
8.5.2 ccgsWalker.g ..... - 69 -
8.6 Types ..... - 73 -
8.6.1 ExpressionList.java ..... - 73 -
8.6.2 Location.java ..... - 74 -
8.6.3 NumType.java ..... - 74 -
8.6.4 StringType.java ..... - 78 -
8.6.5 DataType.java ..... - 81 -
8.6.6 LineLocation.java ..... - 82 -
8.6.7 MatchLocation.java ..... - 85 -
8.6.8 ParamList.java ..... - 88 -
8.7 Interpreter ..... - 88 -
8.7.1 ccgsFunction.java ..... - 88 -
8.7.2 GsccCodeException.java ..... - 89 -
8.7.3 InterpreterImpl.java ..... - 90 -
8.7.4 Locations.java ..... - 102 -
8.7.5 Utils.java ..... - 110 -
8.7.6 Interpreter.java ..... - 110 -
8.7.7 LineReader.java ..... - 112 -
8.7.8 SymbolTable.java ..... - 114 -
8.8 Testing Files ..... - 115 -

1 Introduction: The Proposal

1.1 An Introduction to gscc

gscc is a simple text processing language which provides facilities to execute a variety of rich text manipulation commands. It combines the utility of a number of everyday Unix command-line tools with a simple syntax.

1.2 Background

The wide-range usage of computers in all industries has necessitated employing administrators who are skilled in maintaining the systems on which valuable work is done. Typically, the configuration of these systems relies on complex text files, which must often be manipulated by the administrators. With a computer at their disposal, the administrators created ways to make the machine perform the complex pattern-matching and text-replacement operations.

As computers have been in place for several decades, the needs and abilities of users have increased. Today, very complex and useful systems are maintained in text files: statistics, programming language code, academic papers, news articles, address books, etc. Facilities must exist to modify these types of files based on a set of rules, defined by the user, and yet still be easily accessible to the user.

1.3 Goals of gscc

gscc seeks to be a programming language that fills this niche. It will provide the user a small set of very powerful utilities to allow complex manipulations of their text files.

1.3.1 Simplicity

gscc should be accessible to a wide-range of people with varying computer skills. Therefore, gscc uses simple English commands to manipulate text in complex ways.

1.3.2 Streamlined Operation

gscc iterates over each line and performs all defined commands on that line sequentially, only passing through each file once.

1.3.3 Portability

Since gscc is implemented in Java and uses the Java Virtual Machine to operate, gscc is by definition platform independent. The user is able to manipulate text the same way on a Windows platform as he is a Unix platform.

1.4 Language Features

1.4.1 Small, powerful command set

gscc will present a small number of commands to the programmer that will allow for huge possibilities of utility. The commands allow the programmer to perform the few basic text manipulations: insertion, deletion, appending, and searching. The programmer can perform these operations based on regular expressions and other logic-based decisions.

1.4.2 User-defined subroutines

As text processing requirements vary, the user will be able to specify domain-specific subroutines to acquire the desired result. The user can extend the built-in capabilities of the language by creating subroutines. These subroutines also allow for high amounts of code reuse and simplify potentially-complex code.

1.4.3 Unicode Support

As with international business, the need for manipulating foreign character sets continually increases. By virtue of being based on Java technology, gscc supports Unicode characters and regular expressions.

1.5 gscc in Practice

1.5.1 A First Syntax Example

The following example replaces all instances of “Hello” or “hello” with “Goodbye!” The last line always prints

```
set $bye = "Goodbye!";
[[Hh\]ello] global {
        set @match, $bye;
}
[.*] line { print @line; }
```

1.5.2 Further Examples

The following example prepends all lines that match this regular expression with the line length, but deletes them if they don't begin with A.

```
[t?[a-z\]*66] line {
        if( substr(@line, 0, 1) != "A") {
              delete @line;
        }
         else {
                insert @line, 0, @line.length;
        }
}
[.*] line { print @line;}
```

Imagine a file of grades, where you want to replace numbers 90 through 93 with A-:

```
[ 9[0-3\]] global {
        set @match, "A-";
}
[.*] line { print @line;}
```

…except you have a problem: the University's mainframe system was written in FORTRAN and can't handle last and first names longer than 15 characters. You want to truncate each tuple to "lastname firstname grade".

```
[^[A-Za-z]] global {
        set @match, $substring(@match, 0, 15);
}
[ 9[0-3\]] global { set @match, "A-"; }
[ 9[4-6\]] global { set @match, "A"; }
//continues..
[.*] line { print @line; }
```

2 Language Tutorial

2.1 Compiling/Running gscc

Gscc requires a Java 5 runtime environment.

```
$ cd gscc_language
$ ./compile_gscc
```

This will properly compile the source in gscc_language into class files. You can then run the gscc program on some input file by using either:

```
$ ./run_gscc <program file> <input file>
                    -OR-
$ java –cp.:antlr.jar gscc <program file> <input file>*
```

If you aren't using the script to run gscc, multiple space separated input files can be listed and the program will be run on each sequentially.

2.2 Creating a gscc program

Any text editor can be used to write a gscc program. Files must end with the .gscc extension. Meanwhile, the input files can be any type of file. We are now ready to write out first gscc program:

```
1| /* Filename: hello.gscc */
2| func $foo() {
3|         return "Hello World\n";
4| }
5|
6| [H*] line {
7|         print $foo();
8| }
```

Line 1 has a C style comment that tells us the name of the file.

Lines 2-4 defines the string function $foo() that simply returns the string

```
1| Hello World
```

Lines 6-8 has a line type regexp block that is run once per line every time the regular expression H* (* being the Kleene Star) is matched within a line of the input file.

Depending upon the input that this program is run on, the number of times "Hello World\n" is printed will differ. For example, if the input contained the single line:

```
1| Hello World
```

The resulting output would be:

```
1| Hello World
2|
```

Because the H* was matched to the H in Hello. Now if this was run on the input:

```
1| Hello
2| World
```

The resulting output would be:

```
1| Hello World
2| Hello World
3|
```

As expected because H* would match ever line simply because it count 0 or more occurrences of the letter H. Since there are 2 lines in the input, the regexp block was matched and executed twice.

2.3 Another Example: File Searching

As we all know, the standard hello world example is never very interesting and doesn't employ much of what makes our language special. So, in out next example, we will demonstrate a much broader range of statements that can be used as well as the two special variables @match and @line.

Say we have a specific input file format:

<first name>\t<last name>\t<phone number>

```
    1| John       Doe       222-555-5555
    2| Jane       Doe       333-555-5555
    3|  …
65535| Jonathan   Doe       555-555-5555
```

We want a program that can search for someone by last name and print out the names and numbers that match that specification.

```
1| /* Filename: search.gscc */
2|
3| [\tDoe\t] line {
4|   print @line;
5| }
```

This program seems very simple and yet accomplishes the task. By using the location operator @line, once a matching last name is found in the line, we simply print out the entire line of information. Now let's say we want a more robust program that

searches for information in this file based on multiple requirements. Say we want all people with first name John and whose phone number ends in -4321.

```
 1| /* Filename: complex_search.gscc */
 2|
 3| set #check, 0;
 4|
 5| func #match(#comp) {
 6|  if (#comp == 11)
 7|                 return 1;
 8|  return 0;
 9| }
10|
11| [John\t] line {
12|           set #check, #check+1;
13| }
14|
15| [-4321] line {
16|           set #check, #check + 10;
17| }
18|
19| [] line {
20|           if (#match(#check) == 1)
21|                print @line;
22|           set #check, 0;
23| }
```

By using a global variable in #check, we can add 1 to it if a first name match is found by the regexp on lines 11-13. Then we can add 10 to it if the ending phone number digits match in lines 15-17. The last regexp block (19-23) is called once for every line since it is simply matching nothing. This block will use the function #match() to check if both conditions were met, and if they were, the line we have been checking will be printed out. Then check will be reset to 0 for the next line on the stack.

2.4 Search and Replacement Program

In this next example, we will take the search program and take it one step further. Using the same input, we want to search for people and instead of printing out the data listed, we want to print replace their phone number with a newer correct number and print out that data. For this, we want to update Jane Doe and Jonathan Doe's phone numbers. We then want to print out everyone's information again.

```
1| /* Filename: search_replace.gscc */
2|
3| [Jane\tDoe] line {
4|    replace @line, @match.end + 1, "321-123-4343";
```

```
 5| }
 6|
 7| [Jonathan\tDoe] line {
 8|    replace @line, @match.end + 1, “545-444-1212”;
 9| }
10|
11| [] line {
12|    print @line;
13| }
```

We aren’t limited to just replacements though. We can also delete data if needed as well as insert data that is missing to be printed out. Say we want to delete the line that has John Doe since he has moved away and add in a line for a new resident Agent Smith.

```
 1| /* Filename: remove_insert.gscc */
 2|
 3| set $added, “no”;
 4|
 5| [John\tDoe] line {
 6|    delete @line, @line.start, @line.end;
 7| }
 8|
 9| [\n] line {
10|    if ($added == “no”)
11|    {
12|                set $data, “\nAgent\tSmith\t232-435-3467”;
13|                insert @line, @match.start, $data;
14|                set $added, “yes”;
15|    }
16| }
17|
18| [\t] global {
19|    replace @line, @match.start, “ “;
20| }
21|
22| [] line {
23|    print @line;
24| }
```

By using the delete command on line 6, we reduced the line to nothing and thus, when we go to print it out in the final block (18-20), nothing is printed out, effectively removing it from the file. Meanwhile, at any of the lines, insert simply placed the new data into the @line variable since changes made in that carries over to the subsequent regexp blocks. Thus, when it reaches 18-20, it will print out all of the data including the new data.

Finally in one new addition, we use a global regexp to replace all tab characters with spaces so the data will look more readable when printed out. Global regexps execute once per every match that occurs in the line. Note how since the Agent Smith insertion occurs before the global regexp block, its tabs will also be edited to spaces. This again shows how a change in @line carries over to subsequent blocks.

2.5 A more complicated useful example (Apache log parsing)

The previous example is useful for the specific case listed, but we won't always be working with self formatted files. Instead we want to create programs that will be helpful to a broad range of users. A more useful program would work on an input that, although is predefined, is also one that many people will want to draw information from. A good example of this would be writing a program for an Apache log file. In this final example, we want to find the 404's inside an apache log file and print out the relevant data.

The format of a 404 in an apache log file looks like the following (one line):

221.116.200.62 - - [19/Dec/2005:17:08:36 -0500] "POST /xmlsrv/xmlrpc.php HTTP/1.1" 404 278

Say we want to print out the line of data every time we find a 404. This simple program should suffice by using the built in function $substr() and the location variable @match.

```
1| /* Filename: apache.gscc */
2|
3| [".*"\s404] line {
4|   print $substr(@match, 0, @match.length-4) + "\n";
5| }
```

However, there are times when you want to ignore certain 404's, like when someone is trying to search for vulnerabilities. By utilizing the fact that changes to the special variable @line carry over to the check of the next regexp block, we can make the following changes to ignore those accesses:

```
1| /* Filename: apache_ignore.gscc */
2|
3| [xmlrpc\.php] line { set @line, "";}
4|
5| [".*"\s404] line {
6|   print $substr(@match, 0, @match.length-4) + "\n";
7| }
```

Finally, say we want to take this one step further and print out a running counter that counts the number of 404's that we found in the log file. Again this only takes a minor change in our language by creating a global variable #count.

```
 1| /* Filename: apache_ignore_count.gscc */
 2|
 3| set #count, 0;
 4|
 5| [xmlrpc\.php] line { set @line, "";}
 6|
 7| [".*"\s404] line {
 8|   set #count, #count+1;
 9|   print #count + "\t";
10|   print $substr(@match, 0, @match.length-4) + "\n";
11| }
```

2.6 A really cool example

This next example is an exciting use of our language to do something really cool. Brainf*ck is a computer programming language noted for its extreme minimalism. It was designed to challenge and amuse programmers, and is not suitable for practical use. This is an example of gscc taking brainf*ck code and turning it into something readable, like C. Then you can compile the C and make it run!

First – the sample brainf*ck file. 100 points if you know what this does.

```
head
>+++++++++[<++++++++>-]<.>+++++++[<++++>-]<+.+++++++..+++.[-
]>++++++++[<++++>-]
<.>++++++++++[<+++++>-]<.>++++++++[<+++>-]<.+++.------.--------.[-
]>++++++++[
<++++>-]<+.[-]++++++++++.
end
```

Now, our gscc code to change this mess into something legible.

```
[head] line {
print "#include <stdio.h>\n";
print "#include <stdlib.h>\n";
print "#define INC ++*p;\n";
print "#define DEC --*p;\n";
print "#define SRT p++;\n";
print "#define SLT p--;\n";
print "#define LB while(*p){\n";
print "#define GC *p=getchar();\n";
print "char *p; \n";
print "void *pr(){ putchar(*p);} \n";
print "void main(){  p = malloc(32768); char *q = p; int i; for(
i=0; i<32768; i++){*q=0; ++q;} \n";
```

```
set @match, " ";
}
[end] line {
set @match, "} ";
}

[>] global {
set @match, " SRT ";
}

[<] global {
set @match, " SLT ";
}

[\+] global {
set @match, " INC ";
}

[-] global {
set @match, " DEC ";
}

[\.] global {
set @match, " pr(); ";
}

[\[] global {
set @match, " LB ";
}
[\]] global {
set @match, " } " ;
}
[,] global {
set @match, " GC " ;
}
[.] line {
//insert @line, @line.length-2, "  ";
print @line;

}
```

And its output (in C, of course)

```
#include <stdio.h>
#include <stdlib.h>
#define INC ++*p;
#define DEC --*p;
#define SRT p++;
#define SLT p--;
#define LB while(*p){
#define GC *p=getchar();
char *p;
void *pr(){ putchar(*p);}
```

```
void main(){  p = malloc(32768); char *q = p; int i; for( i=0;
i<32768; i++){*q=0; ++q;}

 SRT  INC  INC  INC  INC  INC  INC  INC  INC  INC  LB  SLT  INC  INC
INC  INC  INC  INC  INC  INC  SRT  DEC  }  SLT  pr();  SRT  INC  INC
INC  INC  INC  INC  INC  LB  SLT  INC  INC  INC  INC  SRT  DEC  }
SLT  INC  pr();  INC  INC  INC  INC  INC  INC  INC  pr();  pr();
INC  INC  INC  pr();  LB  DEC  }  SRT  INC  INC  INC  INC  INC  INC
INC  INC  LB  SLT  INC  INC  INC  INC  SRT  DEC  }
 SLT  pr();  SRT  INC  INC  INC  INC  INC  INC  INC  INC  INC  INC
INC  LB  SLT  INC  INC  INC  INC  INC  SRT  DEC  }  SLT  pr();  SRT
INC  INC  INC  INC  INC  INC  INC  INC  LB  SLT  INC  INC  INC  SRT
DEC  }  SLT  pr();  INC  INC  INC  pr();  DEC  DEC  DEC  DEC  DEC
DEC  pr();  DEC  DEC  DEC  DEC  DEC  DEC  DEC  DEC  pr();  LB  DEC
}  SRT  INC  INC  INC  INC  INC  INC  INC  INC  LB
 SLT  INC  INC  INC  INC  SRT  DEC  }  SLT  INC  pr();  LB  DEC  }
INC  INC  INC  INC  INC  INC  INC  INC  INC  INC  pr();
}
```

Compile this bad boy in gcc and you get the following output:

Hello World!

3 Language Reference Manual

3.1 Introduction

gscc uses simple English commands to manipulate text in complex ways. gscc will present a small number of commands to the programmer that will allow for huge possibilities of utility. The commands allow the programmer to perform the few basic text manipulations: insertion, deletion, appending, and searching. The programmer can perform these operations based on regular expressions and other logic-based decisions.

gscc iterates over each line and performs all defined commands on that line sequentially, only passing through each file once. As text processing requirements vary, the user will be able to specify domain-specific subroutines to acquire the desired result. The user can extend the built-in capabilities of the language by creating subroutines. These subroutines also allow for high amounts of code reuse and simplify complex code.

3.2 Lexical Conventions

3.2.1 Tokens

Tokens define identifiers, keywords, constants, string literals, line terminators, operators, and separators. The entirety of the program is made up of tokens separated by white space. All white space is ignored except when it is used to separate tokens.

3.2.2 *Comments*

Comments can be used through the program to leave notes and/or documentation. They begin with the characters /* and terminate with the characters */. Comments can also start with // and end with a newline. Comments cannot nest.

3.2.3 *Identifiers*

An identifier is a sequence of letters and digits; the underscore character '_' counts as a letter. Upper and lower case letters are different. The first character must be a type prefix of either # or $. Identifiers must have a length of at least 2 characters (including the type prefix). Identifiers cannot be a keyword or the null literal. Identifiers also include the special location variables @match and @line.

3.2.4 *Keywords*

break	if	replace
delete	insert	return
else	length	set
elseif	line	start
end	match	while
func	prerr	
global	print	

3.2.5 *Constants*

3.2.5.1 Integer Constants

An integer constant is a sequence of one or more decimal digits.

3.2.6 *Escape Characters*

In cases when a single character is expected, a string is passed. Constants do not contain newlines; in order to represent these, and other characters, the following escape sequences may be used

Newline	NL(LF)	\n
Horizontal tab	HT	\t
carriage return	CR	\r
double quote	“	\”
backspace	BS	\b
formfeed	FF	\f
backslash	\	\\

3.2.7 White space

Some white space is required to separate adjacent identifiers, keywords, and constants. White space is defined as the ASCII space and horizontal tab characters, as well as line terminators (see below definition).

3.2.8 *String Literals*

A string literal is a sequence of characters surrounded by double quotes. String Literals can contain newline or escaped double quotes.

3.2.9 *NULL Literal*

The null literal is represented by the empty string

3.2.10 *Line Terminator*

Lines are terminated by the ASCII characters CR, or LF, or CR LF. The two characters CR immediately followed by LF are counted as one line terminator, and not two.

3.2.11 *Separators*

The following ASCII characters are the separators

/	)
{	;
}	,
(	.

Operators

The following objects represent operators:

!	]
*	<=
/	>=
+	= =
–	! =
&&	<
||	>
[	

3.3 Error Handling

When an element of the program encounters something that it does not know how to handle, like an improper statement, or an assignment of a mismatched type, or similar; an error occurs. Errors halt execution. Errors occur at either parse time or run time and cause the program to halt.

3.4 Identifiers

Similar to variables in other languages, identifiers are used to store information in gscc. There are only three types of identifiers in gscc: strings, integers, and locations. Identifiers do not need to be formally declared by the user. Instead, upon being called by a system command, such as ‘set’, the identifier will be implicitly declared. If an identifier is referenced and no value was previously defined to it by the ‘set’ system call, then it will be created, initialized with the null string and processing will continue.

The three types of identifiers can be broken down further into basic variables and system variables. Strings and integers fall under the basic variables category while locations are considered system variables. The difference between the two is that the former can have their initial value set by the user, whereas the latter has its value predefined by the interpreter. In addition to this, any number of basic variables can be created for use by the user while the number of system variables is predetermined.

3.4.1 Basic Variables

The two basic variables are strings, denoted by $, and integers, denoted by #. Strings will take on the string value of whatever expression is set to it, while integers take on the integer value of the expression set to it. With automatic conversion, strings will take on the string value of any number set to it, but for integers, the expression set to it must, overall, be an integer value or else an error is thrown. Errors halt execution.

Type: STRING
Syntax: $<token>
Examples:

```
/* this will store the value "Four score and
seven years ago" into the identifier $text */

        set $text, "Four score and seven
        years ago";


/* this will automatically convert the resulting
integer expression into the string "12345" and
store it into $count */

        set $count, 12000 + 345;
```

Type: INTEGER
Syntax: #<token>
Examples:

```
// sets the value in #three to 3
        set #three, 3;

/* since the overall expression of #three + 4 is
an integer, this set is valid and will store 7 into
#seven */

        set #seven, #three + 4;
```

3.4.2 *System Variables*

All system variables are of a special type known as a location, starting with a given prefix @. Locations are set by the interpreter, and represent a location within a given input. They cannot be declared by the user and there are only two location variables: @match and @line.

Name: @match
Description: The string that matched the regular expression.

Name: @line
Description: The line of text containing that string.

Location attributes:
Locations have a few attributes of type integer:

```
@<loc_var>.line    /* Line number
@<loc_var>.start   /* starting column/character, inclusive
@<loc_var>.end     /* ending column/character, inclusive
@<loc_var>.length  /* length of the string value of location
```

Context: When a location is passed to an action or function that expects a string, it is converted to a string automatically by the parser.

3.5 Expressions

3.5.1 *Definition*

An expression (denoted by <expression>) is any group of identifiers separated by operators that are to be evaluated by the interpreter and whose resulting value is to be used by a command, statement, or function. The overall value of an expression depends upon its expected value and also by its components. A combination of strings and/or integers will be evaluated to a string result, while a combination of all integers will be evaluated to either an integer result (if the call is expecting an integer) or a string result (if the call is expecting a string).

3.5.2 *Comma Separator*

Syntax: <expression> , <expression>
Operator: ,
Description: The comma operator is a separator in lists of function arguments and therefore is required to be in a parenthetical grouping such as: func #myhouse(#var, $string)

3.5.3 *Evaluation Order*

All the expressions are grouped and evaluated left to right.

3.5.4 Precedence

The precedence of the operators is as follows (highest precedence on left/top, lowest on right/bottom).

```
                    !
                  *   /
                  +   -
< <= > >=  == != && || ,
```

3.5.5 Unary Minus Operator

The operand of the unary – operator must be a number, and the result is the negative of the operand. The result of applying the unary minus operator to a signed operand is equivalent to the negative promoted type of the operand. Negative zero is zero.

3.5.6 Logical Not Operator

The logical not operator is denoted by the ! character. It negated the value of the <expression> that immediately follows it. A logical operator returns a numerical expression, namely 1 or 0.

3.5.7 Multiplicative Operators

Syntax: <integer1> * <integer2>
Operator: *
Description: The * operator performs multiplication, resulting in the product of its operands. It multiplies the value of <integer1> by the value of <integer2>. Multiplication is commutative and associative.

Syntax: <integer1> / <integer2>
Operator: /
Description: The / operator performs integer division, resulting in the quotient of its operands. The left-hand operand is the dividend, and the right-hand operand is the divisor. It divides the value of <integer1> by the value of <integer2>.

3.5.8 Additive Operators

Syntax: <integer1> + <integer2> | <string1> + <string2>
Operator: +
Description: The + operator performs addition when applied to two operands of numeric type, and concatenation when applied to two operands of type string. Numeric addition is commutative and associative.

Syntax: <integer1> - <integer2>
Operator: -
Description: The – operator performs subtraction, resulting in the difference of the two operands. This subtracts <integer2> from <integer1>. Numeric subtraction is commutative and associative.

3.5.9 Relational Operators

Syntax: <expression> < <expression>
Operator: <
Description: The less than relational operator yields 0 if the relation is false and 1 if it is true. The type of the result is number.

Syntax: <expression> > <expression>
Operator: >
Description: The greater than relational operator yields 0 if the relation is false and 1 if it is true. The type of the result is number.

Syntax: <expression> <= <expression>
Operator: <=
Description: The less than or equal relational operator yields 0 if the relation is false and 1 if it is true. The type of the result is number.

Syntax: <expression> >= <expression>
Operator: >=
Description: The greater than or equal relational operator yields 0 if the relation is false and 1 if it is true. The type of the result is number.

3.5.10 Equality Operators

Syntax: <expression> = = <expression>
Operator: = =
Description: The equals relational operator yields 0 if the relation is false and 1 if it is true. The type of the result is number.

Syntax: <expression> != <expression>
Operator: **!=**
Description: The not equals relational operator yields 0 if the relation is false and 1 if it is true. The type of the result is number.

3.5.11 Logical Operators

Syntax: <expression1> && <expression2>
Operator: &&
Description: The logical and operator yields 0 (false) if either of the Boolean values of <expression1> and <expression2> is false. It yields 1 (true) if both <expression1> and <expression2> yield Boolean values of true.

Syntax: <expression1> || <expression2>
Operator: ||
Description: The logical or operator yields 0 (false) if both <expression1> and <expression2> have Boolean values of false. It yields 1 (true) if either <expression1> or <expression2> are true.

3.6 Syntax Notation

A gscc program is broken up into any number of global variable definitions, function blocks, and regular expression blocks. Global variable definitions are “set” system commands used to associate a value to a variable which will be referenced as a global variable. Function blocks (See Section “Functions”) are blocks of commands and statements grouped together by the user for use throughout the program. Regular expression blocks (See Section “Regular Expression Blocks”) are the main way to execute commands and statements.

A block is a sequence of statements, system commands, and function calls contained within a set of open and close brackets. Blocks are used in the syntax of regular expressions, functions, and statements. The structure of a gscc program is mainly determined by these blocks as defined below.

Program structure:
[global variable definitions]*
[function declarators/blocks]*
<regular expression block>*

3.7 Regular Expression Blocks

Regular expression blocks are the basic structure used in gscc to determine which commands are to be executed and when they are to be executed. Firstly, the 'global' or 'line' keyword determines whether the commands are called once per match or once per match per line. This is similar to the global switch in SED.

If the line is modified, this modification carries through to the subsequent regular expression blocks. This modification may introduce a newline, but it will not affect execution—the line variable will, essentially, contain two lines.

3.7.1 regexp Blocks

Name: REGEXP
Type: GLOBAL
Syntax: [<regexp>] global {
[command | statement]*;
}

Description: For a global regular expression block, the commands and statements inside the brackets are executed once per match. In other words, if the regular expression occurs three times in one line, this block is executed three times for that line. Since a block has the form [<regexp>] line { ... }, the <regexp> must escape all right braces that are part of the regexp. [[^a]] is invalid. [[^a\]] is valid.

Example:

```
/* will run through the commands for every “at” in the input
[at] global {
        set $hi, “hi”;
        set #boo, “33”;
        print $hi + #boo + @match;
}
```

Name: REGEXP
Type: LINE
Syntax: [<regexp>] line {
[command | statement]*;
}

Description: Line regular expression blocks will execute once per every line that contains the regular expression. So, unlike the global

type, if the regular expression occurs three times in the same line, this block is only executed once.

Example:

```
/* will run through the commands once no matter how many
times "at" occurs in the same line */
        [at] line {
                set $hi, "hi";
                set #boo, "33";
                print $hi + #boo + @match;
        }
```

3.7.2 Regular Expression

Since this program is based on searching for the regular expression within the input, it is critical that we define a regular expression. The definition used is the same as the one presented in Michael Sipser's book: *Introduction to the Theory of Computation*.

Say that R is a regular expression (<regexp>) if R is:

1. *a* for some *a* in the alphabet ∑,
2. *ε*,
3. *Ø*,
4. (R_1 ∪ R_2), where R_1 and R_2 are regular expressions,
5. (R_1 ○ R_2), where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

3.7.3 Most Commonly used Regular Expressions

gscc uses the Java 1.5 implementation of regular expressions. The explicit details for the regexp constructs can be found at http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html. Below we have listed some of the more common regular expression constructs that can be used with gscc.

The backslash character (`'\'`) serves to introduce escaped constructs, as defined in the table above, as well as to quote characters that otherwise would be interpreted as unescaped constructs. Thus the expression `\\` matches a single backslash and `\{` matches a left brace.

XY	*X followed by Y*
X|Y	*Either X or Y*
^	*The beginning of a line*
$	*The end of a line*
[abc]	*a, b, or c (simple class)*
[^abc]	*Any character except a, b, or c (negation)*
*X**	*X*, zero or more times
X+	*X*, one or more times

$X\{n\}$ X, exactly n times

3.8 Statements

There are only three available statement types for use in gscc: selection, iteration, and return. These statements are similar to those in C, C++ and Java in both syntax and usage. Few statements are provided, but due to their robustness, they are able to accomplish the same functionality of statements in most other languages.

3.8.1 Selection Statements

Name: IF
Syntax: if(<expression>) {
[command | statement]*;
}
Description: This statement performs a Boolean check on the given <expression>. If it is evaluated to be true, then the commands and statements within the open and closed brackets are executed.
Example:
/* if #three contain the value 3, then the set command will be executed otherwise, this block is skipped */

```
/* will be executed*/
set #three, 3;
if (#three = = 3) {
set $three, “yes”;
}
```

```
/* will not be executed*/
set #three, 4;
if (#three = = 3) {
set $three, “yes”;
}
```

Name: ELSE IF
Syntax: <IF syntax>
else if (<expression>) {
[command | statement]*;
}
Description: This statement is executed if the Boolean check on the preceding IF block results in a false value. It will then perform another Boolean check on the <expression> for the else if statement. If this is evaluated to be true, then the commands and statements within the open and closed brackets are executed.
Example:
/* if #three contain the value 3, then the set command will be executed otherwise, this block is skipped. The else if will be executed if #three contains the value 4 */

```
/* IF will be executed*/
set #three, 3;
if (#three = = 3) {
set $three, “yes”;
}
else if (#three = = 4) {
set $three, “no”;
}
```

```
/* ELSE IF will be executed*/
set #three, 4;
if (#three = = 3) {
set $three, “yes”;
}
else if (#three = = 4) {
set $three, “no”;
}
```

Name: ELSE

Syntax:
```
<IF syntax>
[ELSE IF syntax]*
else {
        [command | statement]*;
}
```

Description: This statement is executed if the Boolean check on all of the preceding IF and ELSE IF block statements results in a false value. No check is needed for this statement; instead, if executed, it will execute all the commands and statements within the open and closed brackets.

Example:

/* if #three contain the value 3, then the set command will be executed otherwise, this block is skipped. The else if will be executed if #three contains the value 4. If neither are true, then the else is executed */

```
/* IF executed*/
set #three, 3;
if (#three = = 3) {
set $three, “yes”;
}
else if (#three = = 4) {
set $three, “no”;
}
else {
set $three, “ack”;
}
```

```
/* ELSE IF executed*/
set #three, 4;
if (#three = = 3) {
set $three, “yes”;
}
else if (#three = = 4) {
set $three, “no”;
}
else {
set $three, “ack”;
}
```

```
/*ELSE executed*/
set #three, 5;
if (#three = = 3) {
set $three, “yes”;
}
else if (#three = = 4) {
set $three, “no”;
}
else {
set $three, “ack”;
}
```

3.8.2 Iteration Statements

Name: WHILE

Syntax: while(<expression>) {
[command | statement]*;
}

Description: This statement will execute the commands and statements inside the open and closed brackets while the <expression> evaluates to true under a Boolean check. The expression is checked when the statement is first encountered, and then checked again each time the program reaches the closed bracket. This loop will continue to repeat until the <expression> evaluates to be false; it will then skip over the block and precede normally with the program.

Example:

/* this will loop and keep performing the set commands until #boo is greater than 3 */

```
/* final value of $boo is 4*/
set #boo, 1;
while (#boo <= 3) {
set #boo, #boo + 1;
set $boo, #boo;
}
```

```
/* skips the while entirely*/
set #boo, 5;
while (#boo <= 3) {
set #boo, #boo + 1;
set $boo, #boo;
}
```

3.8.3 Return Statement

Name: RETURN

Syntax: return <expression>;

Description: The return statement is used in a function to return the appropriate value back to where it was called. If expecting a string($) return, then the <expression> returned will automatically be converted to a string. If an integer(#) is expected, then <expression> should evaluate to an integer value or an error will be thrown. As previously stated, a error halts execution. When a return is encountered in a function block, it will immediately break out of the function after the return is executed.

Example:

```
/* at the end of the function, the value of $hello is returned*/
        func $helloworld (#world) {
                set $hello, #world;
                return $hello;
        }
```

3.9 System Commands

System commands are the only ways to perform an operation on an identifier in gscc. All system commands are predefined by the compiler and cannot be created by

the programmer. There are only two types of system commands: those that perform operations on identifiers and those that print expressions (either to stdout or stderr). They all take this general form:

<system command> <comma separated list of identifiers and/or expressions>;

System commands do not return any value unlike functions (See Section "Functions"). Operational types will perform an operation on the first argument based to the command called and the following parameters given to it. Printing types, on the other hand, consist of simply the command and the expression that is to be printed. All system commands are zero-based.

3.9.1 Operational System Commands

/* for all of the following definitions:
<string> refers to either a string identifier($) or a location(@)
<integer> refers to either an integer identifier(#) or an actual integer */

System Command: SET
Syntax: set <identifier> , <expression>
Parameters: <identifier> - can be any identifier
<expression> - this can be any expression

Description: This command will replace whatever value that is originally in <identifier> with the value given by <expression>. For locations(@), their string value will be replaced by the value specified in <expression>. Integers(#) must have a corresponding integer valued <expression>, while strings($) will have the <expression> value automatically converted, if necessary.

Examples:

```
/* would replace the location's string value with "abc"*/
        set @match, "abc";

/* would replace the string $myvar with "1234"*/
        set $myvar, "1234";

/* would replace the location's string value by the expression
"russssur" + $myvar, which if using the above set, would give the
overall string value "russssur1234" */
        set @match, "russssur" + $myvar;

/* would replace the location's string value by the expression
@line + $var which takes the string converted value of @line
(let's say this is "4") and concats it to $myvar giving "41234" */
        set @match, @line + $myvar;

/* would replace the integer #mynum with 123*/
```

```
set #mynum, 123;

/* would replace the integer #sub with the expression */
#mynum – 23 which results in 100 */
set #sub, #mynum – 23;
```

System Command: DELETE
Syntax: delete <string> <integer> <integer>
Parameters: <string> - the string from which to delete
<integer> - starting index
<integer> - number of characters to delete
Description: This command will delete characters starting at the index given by the first <integer>. The number of characters deleted will be equal to the number given by the second <integer>.
Examples:

```
/* deletes five characters from index 3 within @match*/
delete @match, 3, 5;


/* deletes 2 characters from index 1 within $hello leaving “hlo”*/
set #one, 1;
set #two, 2;
set $hello, “hello”;
delete $hello, #one, #two;
```

System Command: INSERT
Syntax: insert <string> <integer> <expression>
Parameters: <string> - string where <expression> will be inserted
<integer> - index where insertion will be done
<expression> - the expression that is to be inserted
Description: This command will insert the string value of <expression> into the string <string> at index <integer>.
Examples:

```
/* inserts this string at index 3, if @match was “hihihi”
after insertion, it would contain “hihabcihi” */
insert @match, 3, "abc";

/* inserted the expression @match + “121” into $test
at index 0, giving “hihabcihi121testing123” */
set $test, “testing123”;
set #zero, 0;
insert $test, #zero, @match + “121”;
```

System Command: REPLACE

Syntax: replace <string> <integer> <expression>
Parameters: <string> - the string where replace will be done
<integer> - starting index
<expression> - expression to replace with
Description: Starting at index <integer>, this will replace in <string> the string converted value of <expression> expanding as necessary.
Examples:

```
/* replaces $hi starting at index 0 with “blah” leaving “blahlalalala”*/
        set $hi, “lalalalalala”;
        set #zero, 0;
        replace $hi, #zero, “blah”;

/* given that @match was “hellooooo”, this will begin replacement
at index 5 and expand as necessary leaving “helloabcabcab331” */
        replace @match, 5, "abcabcab331";

/* does replacement at 2 with expression $hi + “11” in @match
resulting in “heblahlalalala31” */
        replace @match, 2, $hi + “11”;
```

3.9.2 Printing System Commands

System Command: PRINT
Syntax: print <expression>
Parameters: <expression> - expression to be printed to stdout
Description: This will print whatever value is in <expression> to stdout.
Examples:

```
/* prints value in $hi to stdout. This is your basic hello world*/
        set $hi, “hello world”;
        print $hi;

/* given that @match is “I am Sam” this prints
“I am Sam, hello world” to stdout */
        print @match + “, “ + $hi;
```

System Command: PRERR
Syntax: prerr <expression>
Parameters: <expression> - expression to be printed to stderr
Description: This will print out the string value of <expression> to stderr.
Examples:

```
/* would print “you messed up” to stderr*/
        prerr "you messed up";
/* will print the expression “you messed up” + $big to
stderr, which is “you messed up big time” */
```

```
set $big, " big time";
prerr "you messed up" + $big;
```

3.10 **Functions**

Functions are named blocks of statements that return one of two types: string or integer. All functions must be declared before any regular expression blocks in the program. Arguments are passed to the function by value. To write a function, the following convention is used:

func <identifier>(<idenitfier list>) { }

1. The func keyword must precede the entire declaration.
2. This is followed by an appropriate identifier as per the function's return type. $<token> identifiers signify functions that return strings while #<token> identifiers signify integer return types.
    2a. The special identifier type @<match|line> cannot be used since function types are restricted to only string and integer types.
    2b. The list of identifiers used for the function names must be unique. Although both #one( ) and $one( ) can co-exist, $two( ) and $two(#two) cannot. In other words, function overloading is not allowed.
3. After the identifier is a comma separated identifier list enclosed by an open and closed parenthesizes. The identifier list can have either many or no parameters.
4. Finally, after the parameter list is all of the statements associated with the function enclosed in open and closed brackets.

The execution of the function begins with the first statement after the opening bracket. It will proceed statement by statement until either a return is executed, or it reaches the closing bracket of the function. If no return call is encountered before the end of the function, then the NULL value is returned by default. It is important to note that for these "pseudo-void" type functions, the user must still decide if it is a string or integer type function.

Example:

```
func $example($one, #two, $three) {
        if (#two > 2) {
                set $a, $one + $three;
                return $a; }
        set $b, "this makes no sense";
        return $b;
        }
```

$example is an example of a string function that will concat the two strings if the parameter #two is greater than 2 and then return that value. However, if #two is not greater than 2, then it will set $b to "this makes no sense" and return that string to where it was called.

Example:

```
func #isTwo($two, #two) {
  if ($two = = “two”) {
            set #add, #two + 2;
            return #add; }
  set #sub, #two – 2;
  return #sub;
  }
```

Meanwhile, #isTwo is an integer type function that will set #add as the value of #two plus 2 and return that value if $two is the string “two”. Otherwise, it will set #sub as the value of #two minus 2 and return that resulting value.

Example:

```
func $expA( ) {
   set #a, @line;
   if (#a = = 1) {
      return “one”; }
   else {
      set $b, "booooo";
      return $b; }
}
```

```
func $expB( ) {
   set #a, @line;
   if (#a = = 1) {
      return “one”; }
   set $b, "booooo";
   return $b;
}
```

In the last example, both are string functions that perform the same procedure: both will return the string “one” if the value in @line turns out to be 1 and return “booooo” otherwise. But upon compilation, the first function is not guaranteed to call a return, but as stated before, a return is not necessary; thus, both functions are still valid.

3.11 Built in functions

There are two built in functions pre-defined in the interpreter for use by the programmer. These functions return a value, and are Since both names are keywords and function overloading is prohibited, these two functions cannot be overwritten.

Function Name:	$substr
Syntax:	$substr($variable, #start, #length)
Parameters:	$variable – represents the string from which the substring will be taken from #start – the starting index #length – the ending index
Description:	The function will return the substring represented by that location from indexes represented by #start to #length inclusive. Since both # and @ identifiers are automatically converted to string

types, you can retrieve the substring of strings, numbers, and locations.

Example:
```
set $test, $substr("abc123", 2, 4);
/* This will set the value in $test to "c12" */
```

Function Name: #length
Syntax: #length($string)
Parameters: $string – the string whose length is to be determined
Description: Will return the length of the given string.
Example:
```
/* This will print '5' to stdout */
set $hello, "hello";
print #length($hello);
```

3.12 Scope

The scope of any identifier can be broken down into either global or local scope. Scope for any identifier is defined at the smallest containing block for that identifier. To clearly explain the scope of an identifier, we will describe lexical, stack based, and nested scope.

1. Lexical Scope

The main concern for lexical scope is the indexing of string($) identifiers. If the programmer attempts to reference any index value outside of the maximum and minimum indexes of the string($), a runtime error will be thrown.

2. Stack Based

Within a given function, there is no access to any identifiers/variables defined outside of the function other than globally defined identifiers/variables. If one is referenced, a null value will be returned.

3. Nested Scope

Any identifiers that are defined outside of all function and regexp blocks are considered to be global identifiers/variables. Identifiers that are defined within a function or regexp block are considered local identifiers in that block, and are inaccessible outside that block. Any identifiers that are defined in a conditional block are local to that conditional block only and die when the conditional block terminates.

4 Project Plan

4.1 Team Responsibilities

Eric:	Frontend and testing
Casey:	Backend and architecture
Russ:	Documentation
Mike:	testing

4.2 Programming Style Guide

4.2.1 ANTLR coding style:

Simple ANTLR rules are written in one line, with sufficient whitespace between characters and logical groups of characters for comprehension. More complex rules are written in multiple lines; the ':' character always aligns with subsequent '|' characters, and also the final ';', itself on its own line.

Where appropriate, additional horizontal space is required to show hierarchies within a single rule, similar to the recommended multi-line regular expression syntax in Perl. Code contained within ANTLR-specified pure Java blocks follow the java programming style.

All lexer names are in uppercase, and all parser and walker rules are in lowercase.

4.2.2 Java coding style:

4.2.2.1 Spacing issues:

For simplicity, we follow the coding style reflected in Eclipse's refactoring tools.

- Indentation at each level is a tab, representing four spaces.
- The initial left brace '{' of a block is the last character of the line before the first line of the block.
- The trailing right brace '}' of a block occupies its own line at the level of the line before the first line of the block.
- There will be no initial or trailing spaces for a parameter list.
- There will be no space between the parameter list and a function name.
- the else keyword cuddles the preceding if block.

4.2.2.2 Nomenclature:

- Classes are placed into three classes according to their utility:
- gsccANTLR, for frontend components, interpreter, for backend components, and types, containing the type framework.
- Variables and function names are in camelcase and should be self- documenting where use is not obvious.

4.3 Project Timeline

ID	Task Name	Ownership	Duration	Start	Finish	Predecessors
1	Choose Teams	All	1 day	Thu 9/8/05	Thu 9/8/05	
2	Select a Meeting Time	All	1 day	Sat 9/10/05	Sat 9/10/05	1
3	Pick A Team Leader	All	1 day?	Sun 9/11/05	Sun 9/11/05	2
4	Install Eclipse	All	3 days	Mon 9/12/05	Wed 9/14/05	1

4.4 Software Project Environment

4.4.1 Operating Systems

Because of Java's "write anywhere, run anywhere" mantra and Perl's aged ubiquity, gscc is able to run on a multitude of platforms simply by relying on a base technology. The authors of gscc simultaneously ran gscc on Windows, Linux, and Mac OS X. There

is nothing preventing our interpreter from running on any other platform that supports these basic tools.

4.4.2 Java 5.0

The vast majority of the code written by our group was in Java. Java's creators at Sun Microsystems describe it as, "Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language." Java was an ideal language for the creation of gscc in that it supports heavy object-orientation and has strong string support. Java 5.0 brought incremental improvement over Java 1.4 for gscc, chiefly in its updated and expanded libraries.

4.4.3 ANTLR

ANTLR is a framework that allowed us to quickly and reliably construct a lexer, parser, and tree walker which would serve as the frontend to our language. From roughly 1500 simple lines of code created in ANTLR's specification language, it creates approximately 4,500 lines of Java to completely describe how to build and walk an abstract syntax tree constructed from the GSCC language.

4.4.4 Subversion (SVN)

Subversion is an up-and-coming replacement to CVS as a source code revision control system. Many of CVS's ailments have been fixed in Subversion, as the new system tracks changes in the file tree as opposed to individual files. This tool allowed us to concurrently work on our source tree and easily transmit changes to the entire group.

4.4.5 Eclipse

Eclipse is an open-source integrated development environment targeted at the Java platform. It has been designed with extensibility in mind, allowing for a vast array of plugins to expand the IDE's already impressive capabilities. The availability of the Subclipse plugin allowed us to seamlessly edit the source tree, commit our changes, and integrate other member's changes into our working copy. The ANTLR plugin for Eclipse provided us with the means not only to compile our grammars within Eclipse with a single click (or every time we saved), but with an integrated means to see and respond to errors in our grammar's code.

4.4.6 Perl

Perl is a widely-used programming languages with applications in small scripts to large web applications. We use a simple Perl script to provide a testing framework for the interpreter.

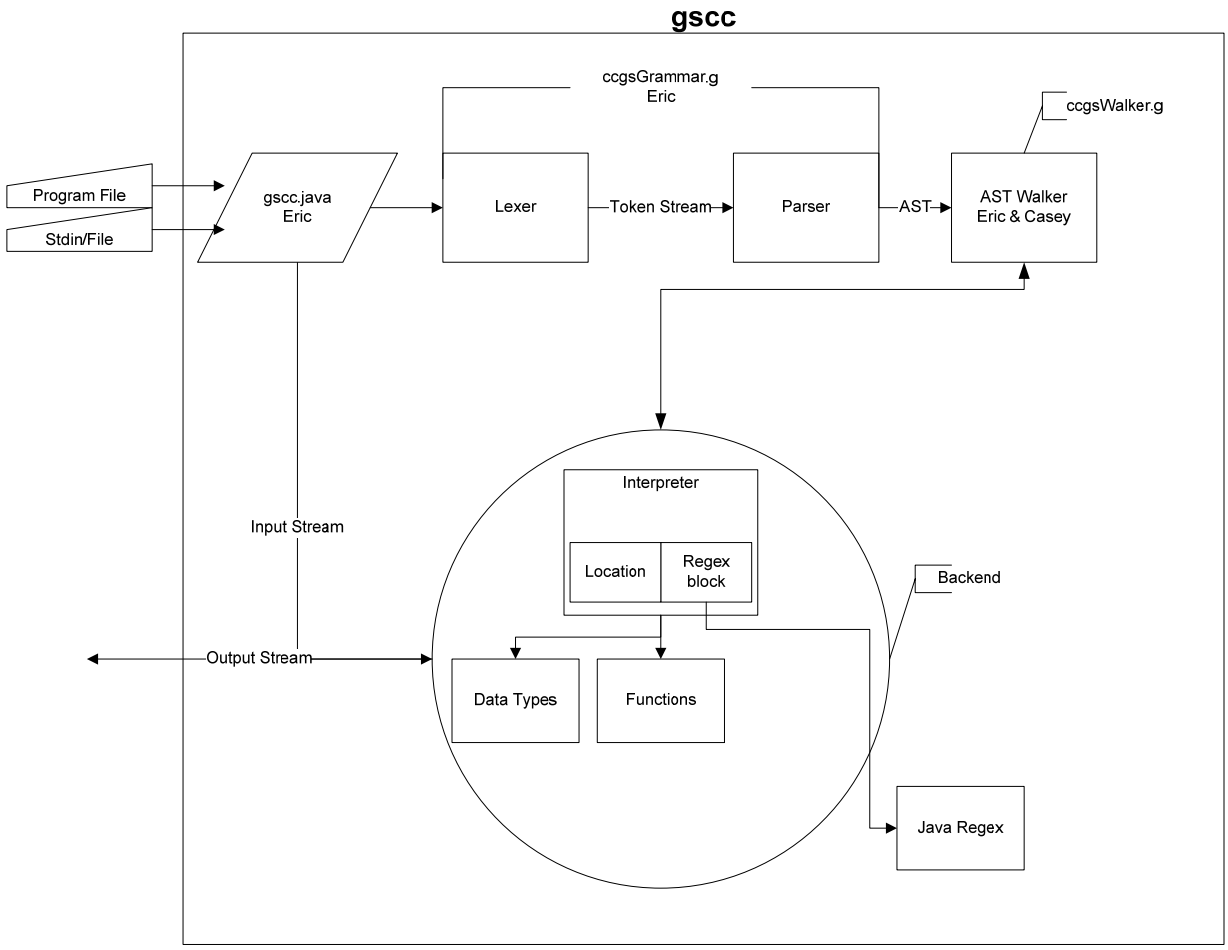
4.5 Project Log

Below are some dates extracted from the SVN logs showing project milestones.

Sept 11	SVN Repository Started
Sept 21	Preliminary Syntax Document
Sept 23	First Draft of Proposal
Sept 25	Final Proposal
Oct 9	Preliminary LRM
Oct 13	Syntax Modification
Oct 18	Grammar Started (oops :-P)
Oct 23	Grammar Finalized
Nov 12	Added Data Types to the Interpreter
Nov 14	Tweaked Tree Walker
Nov 25	Functions Work
Nov 29	Light Testing Begins
Dec 10	Major Coding Complete
Dec 10	Major Testing Begins
Dec 10	First Draft of Final Report
Dec 16	Bug Fixing Complete

5 Architectural Design

5.1 Block Diagram



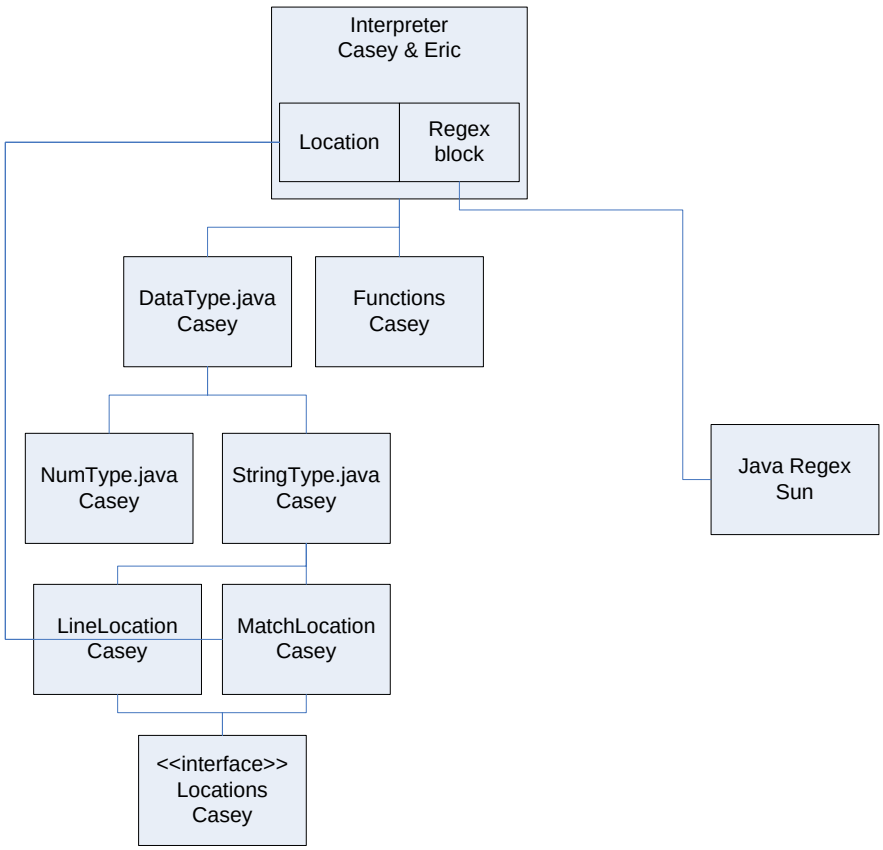
5.2 Component Interfaces

Architecture of gscc:

gscc's architecture can, roughly, be broken in to two separate components. The "front end" consists of code that lexes code, parses that code into a tree, and walks the resulting tree. The front end also includes a small file responsible for initial setup and user interface. The back end is a state machine that handles execution. It is responsible for recognizing regular expressions, storing variables and function definitions, and acting upon commands. This distinction is only approximate, however. In reality, there are many calls from the back-end to the walker.

The main method is included in the file *gscc.java*. It opens source code, and directs the front end to parse and walk the file. The first walk only sets up references within the back end to the overall structural elements of the program. Since a program, at the top

level, can only consist of regular expression bocks, function definitions, and global variable definitions, the back end must store these for later execution.



The back end maintains a symbol table tree and a list of functions and regular expression blocks. For each function and block, it stores the associated statement tree node. It reads input line-by-line, and compares it against successive regular expressions. If input matches, the walker is called to execute beginning with the node previously stored. If, in the course of execution, a function is called, the interpreter handles symbol table modification and directs the walker to the new tree.

There are a number of separate classes within the back end. A number of classes are used to represent the types within gscc. These files are contained in the *types* folder. The superclass, *DataType.java*, is an abstract class with declarations of all the methods that are supported within types. They are implemented in the various subclasses. *NumType.java* and *StringType.java* are the primary types.

In addition, two specialized classes, *MatchLocation.java* and *LineLocation.java* are actually adapters for methods within the interpreter, and allow locations to be treated like a standard type. Within the *interpreter* directory, locations are handled by the L*ocations.java* file. Internally, they are represented by a linked list, and this file implements these operations. *SymbolTable.java* maintains a list of names with their respective values. It also handles the necessary operations required in a statically scoped language.

6 Test Plan

6.1 Test Suites

ADVANCED
BUILTIN
DELETE
ELSE
ELSEIF
FUNCTIONS
IF
INSERT
LENGTH
LOCATION
LOCATIONS
PRINT
REGEX
REGEXP
REPLACE
SET
SUBSTR
WHILE

These test suites each contained many variations of tests. Some were very simple, testing basic functionality while others were complex to test advanced features of out language.

Additionally, Casey and Eric had individual scripts to run while coding; modified according to the feature being implemented. Casey programmed a simple way to graphically see the AST after parsing to visually inspect its consistency. This was utilized by all the testers.

6.2 Reasoning

The purpose of testing is simple: to find out what doesn't work, or what doesn't work right. A good test suite will thoroughly attempt to cover all the categories of features implemented by the language, as well as different levels of complexity and combinations of these features. Our test suite attempts to do this and in turn root out any bugs or features that may have not been correctly implemented initially.

6.3 Automation

The perl script created runs the test programs one at a time. It can run all test in a particular directory, or all of the tests in the test suite. The script can also diff the output

against expected, human generated output contained and compartmentalized into one file for each test.

6.4 Accountability

Mike generated most of the test cases, with Eric and Casey filling in the gaps, correcting them, and correcting the output files, where necessary. Eric wrote the initial testing script, Casey enhanced it with being able to run one script and specifying the text file to run it against, Eric improved it to add diff capabilities.

6.5 Examples

The following examples are given to try and illustrate the thoroughness of testing that we attempted to accomplish. The first test case is a simple one testing the IF conditional and producing a small output. The second case is much more involved and tests a good deal of cases of comparisons that can be done in our language, some simple as 1 != 2 but as complicated as string comparisons. All of out test suites have this variety in test cases to make sure we were thorough in implementation.

6.5.1 Input file -- Mary.txt

```
Mary had a little lamb
With fur as white as snow.
```

6.5.2 Test 1 -- if_test1.gscc

```
/*
Use input from file Mary.txt
all of the if tests will also be testing for the functionality
of the different operators <, >, <=, >=, ==, !=, &&, and ||
*/

[Mary] line {
    if (1)
    {
                set $hello, "hello";
                print $hello + "\n";
    }
    set #num, 1;
    if (#num)
    {
                set $hello, "hi";
                print $hello + "\n";
    }
}
```

```
[snow] line {
    if (0)
    {
                set $hello, "hi";
                print $hello + "\n";
    }
    set #number, 0;
    if (#number)
    {
                set $hello, "hello";
                print $hello + "\n";
    }
}

//I'm predicting this output
/*
hello
hi
*/
```

6.5.3 Test 2 -- if_test2.gscc

```
 /*
if tests, all will be done on the input from file Mary.txt
*/

[Mary] line {
    set $test, "this is a sentence\n";
    set $n, 5;

    if($n){
                print "YES single number variable in if works\n";
    } else {
                print "NO single number variable in if does not work\n";
    }


    if(1){
                print "YES single number in if works\n";
    } else {
                        print "NO single number in if\n";
    }

    if(0){
                print "NO else doesn't work\n";
```

```
} else {
        print "YES else works\n";
}

if(0){
                print "NO elseif doesn't work\n";
} else if (1) {
        print "YES elseif works\n";
}

if($test){
        print "YES single string in if works \n";
} else{
        print "NO single string in if doesn't work \n";
}

set $test2, "";

if($test2){
        print "NO single empty string in if doesn't work\n";
} else {
        print "YES single empty string in if does work\n";
}

if(-1){
        print "YES negative in if does work\n";
} else {
        print "NO negative in if doesn't work\n";
}

if(1+1){
        print "YES expression in if works\n";
} else {
        print "NO expression in if doesn't work\n";
}

if(3 > 2){
        print "YES gt works\n";
} else {
        print "NO gt doesn't work\n";
}

if(2 < 3){
        print "YES lt than works\n";
} else {
        print "NO lt doesn't work\n";
```

```
}

if(2 <= 3){
        print "YES lteq works\n";
} else {
        print "NO lteq doesn't work\n";
}

if(2 <= 2){
        print "YES lteq works\n";
} else {
        print "NO lteq doesn't work\n";
}

if(3 >= 2){
        print "YES gteq works\n";
} else {
        print "NO gteq doesn't work\n";
}

if(3 >= 2){
        print "YES gteq works\n";
} else {
        print "NO gteq doesn't work\n";
}

if(2 == 2){
        print "YES eq works\n";
} else {
        print "NO eq doesn't work\n";
}

if(2 != 2){
        print "NO neq doesn't works\n";
} else {
        print "YES neq does work\n";
}

if(1 != 2){
        print "YES neq\n";
} else {
        print "NO neq \n";
}

if((1==1) && (2==2)){
        print "YES && \n";
```

```
} else {
        print "NO && \n";
}

if((0==1) && (2==2)){
        print "NO && \n";
} else {
        print "YES && \n";
}

if((0==1) && (1==2)){
        print "NO && \n";
} else {
        print "YES && \n";
}

if((0==1) || (2==2)){
        print "YES || \n";
} else {
        print "NO || \n";
}

if((1==1) || (1==2)){
        print "YES || \n";
} else {
        print "NO || \n";
}

if((1==2) || (1==2)){
        print "NO || \n";
        print "YES || \n";
}

set #h, 1 > 2;
if(#h == 1){
```

```
} else {
        print "NO value returned from gt \n";
}

set #h, 1 < 2;
if(#h == 1){
        print "YES value returned from lt \n";
} else  if(#h == 0){
        print "NO value returned from lt \n";
} else{
        print "NO value returned from lt \n";
}

set #h, 3 < 2;
if(#h == 1){
        print "NO value returned from lt \n";
} else if(#h == 0) {
        print "YES value returned from lt \n";
} else{
        print "NO value returned from gt \n";
}

set #h, 1;

if(! #h) {
        print "NO value returned from ! \n";
} else {
        print "YES value returned from ! \n";
}

if( (0 || 0) && 1){
        print "NO (0 || 0) && 1 \n";
} else {
        print "YES (0 || 0) && 1 \n";
}

if( (0 || 1) && 1){
        print "YES (0 || 1) && 1 \n";
} else {
        print "NO (0 || 1) && 1 \n";
}

if( 0 || 0 && 1){
        print "NO 0 || 0 && 1 \n";
} else {
        print "YES 0 || 0 && 1 \n";
```

```
    }

    if( 0 && 0 && 1){
            print "NO  0 && 0 && 1 \n";
    } else {
            print "YES  0 && 0 && 1 \n";
    }

    if( 1 && 1 && 1){
            print "YES  1 && 1 && 1 \n";
    } else {
            print "NO  1 && 1 && 1 \n";
    }

    if( 1 || 0 || 0){
            print "YES   1 || 0 || 0 \n";
    } else {
            print "NO   1 || 0 || 0 \n";
    }

    if( 0 || 0 || 0){
            print "NO 0 || 0 || 0 \n";
    } else {
            print "YES 0 || 0 || 0 \n";
    }
}
```

7 Lessons Learned

- Democratic programming teams work only when each member of the team is focused and goal-oriented. A team leader should have been appointed early on, as instructed, and made a focus of his work managing the team. The team leader should provide motivation, planning, foresight, and observe progress. He should have set goals with deadlines and made it his job to facilitate communications between the team members. The democratic team is a great idea with the capacity for huge amounts of productivity, in theory, but requires the right circumstances.

- Constant communication beyond team meetings can help to keep things flowing. If any of the members isn't performing for whatever reason, having people there to remind them serves as a good motivating factor. It can also help to clear up any confusion that may exist on where the project is headed. Any members who aren't performing up to par should be dealt with early on instead of later. Communication is never easy when it's just over email. Working near a partner encourages you to stay focused on the project.

- Deadlines are an important thing to both know and create. Knowing when what is due keeps people on track and will prevent any unforeseen mishaps. They can also serve as a way to enforce team members to submit work if needed.

- When you are a developer, you want to only think about writing code. Choice of environment is key. We were able to develop very efficiently because we put a lot of thought into our environment and back-end before we even started. An auto-merge file repository meant we never had to worry about losing files.

- However, easily the most important lesson learned was never compromise on your environment. Spending a few hours setting it up in the beginning is easily the best thing you can do with your time. We created our own SVN server, with automated nightly backups. Eclipse had plugins for both ANTLR and SVN. Furthermore, this platform reduced the learning curve of each of these tools as the code administration such as commenting, project to-do lists, bug tracking, and versioning systems were relegated to easy to use user-interface elements. Performing these tasks using CVS, Emacs, and the command- line are is a known-good platform of coding, but provides far less capability.

- Iterative and evolutionary design are keys to developing good ideas. First, an initial idea evolves from ideas that were conceived before and well-known. That idea is transformed into something entirely new by iteratively looking at each level and asking what can be changed to make it more interesting. Once a design is finalized, it evolves into something slightly different once unforeseen challenges arise and problems are solved.

- I learned the importance of a good interface. The developers were able to separate their efforts because the interfaces were well defined. Eric worked on the walker, and I was free to change the interpreter all I wanted, as long as the functions complied.

- Start early, Start early, Start early. There is no better feeling in the world than finishing your duties or a project ahead of schedule. There is no worse feeling than missing a hard deadline.

- If you need to know something don't be afraid to ask. A myriad of stupid questions in rapid succession is bound to get you a better answer than no questions at all. There is more than one person in this group for a reason. If you don't know the answer chances are someone else in your group will or will at the least be able to point you in the right direction. Keep asking until you get the answer you want. If your group dynamic is good, this should be easy.

For other teams, we recommend spending a few hours and really get to know these tools:

http://www.eclipse.org -- Eclipse IDE
http://ANTLReclipse.sourceforge.net/ -- ANTLR plugin for eclipse
http://subversion.tigris.org/ -- Subversion version control system
http://subclipse.tigris.org/ -- Eclipse SVN plugin
http://e-p-i-c.sourceforge.net/ -- Eclipse PERL plugin
http://www.apple.com/macosx/ -- The best development platform there is

8 Complete Listing

8.1 Input Files

8.1.1 Mary.txt

```
Mary had a little lamb
With fur as white as snow.
```

8.1.2 helloworld.bf

```
head
>+++++++++[<++++++++>-]<.>+++++++[<++++>-]<+.+++++++..+++.[-
]>++++++++[<++++>-]
<.>++++++++++[<+++++>-]<.>++++++++[<+++>-]<.+++.------.--------.[-
]>++++++++[
<++++>-]<+.[-]++++++++++.
end
```

8.2 Script Files

8.2.1 compile_gscc

```
#!/bin/bash

svn up
cd gsccAntlr
java -cp ../antlr.jar antlr.Tool ccgsGrammar.g
java -cp ../antlr.jar antlr.Tool ccgsWalker.g
cd ..
javac -cp .:antlr.jar gscc.java
```

8.2.2 run_gscc

```
#!/bin/bash

java -cp .:antlr.jar gscc $1 $2
```

8.2.3 clean_gscc

```
#!/bin/bash
rm -rf *.class
```

8.3 Example Files

8.3.1 simple.gscc

```
[\+] global {
        set @match, "p++; ";
        print @line;
}
```

8.3.2 test.gscc

```
/*
        this is a comment
*/
//this too is a comment
//this is a second comment
func #simpleFunc ($a, $b, #x, #y){
    return 0;
}

func #add(#a, #b){
        while(#a){
                break;
        }
    return #a + #b;
}

func #arith(#a, #b){
        set #c, #a+#b;
        set #c, #a-#b;
        set #c, #a / #b;
        set #c, #a*#b;
}

[anyt\]hing] global {
        print @match.start;

        set $x, "variable x\n";
        set #y, 5;
        #arith($x, $x, #y, #y );
        set $y, $x;
        print $x;
        print $y;

        set $x, "9";
        set $y, $x + "9 bottles";
    print $y;

    setchar $y, 0, "8";

    print $y;

    set @match, "anything was found";
```

```
        /*print @match.line + @match.start * @match.end / @match.length;*/
        /*set @match, "line: " + @match.start + @match;*/
        print @match.line;
}

[any other thing] line {

        set #n, 0;
        while(#n < 10){
                print #n * 10;
                set #n, #n + 1;
        }

        set #n, 1;

/* prints true */
           if( #n > 8 ) {
               print "true";
           } else {
             prerr "false";
           }

/* prints true */
           if( #n * 3 ) {
               print "true";
           } else {
             prerr "false";
           }

/* prints false */
           set #n, 0;
           if( #n ) {
               print "true";
           } else {
             prerr "false";
            }

/* prints true */
            if( #n + 1 ) {
               print "true";
            } else {
               prerr "false";
            }

/* prints true */
            if( #n == 0 ) {
               print "true";
            } else {
               prerr "false";
            }

/* prints false */
            if( #n > 0 ) {
               print "true";
            } else {
               prerr "false";
```

```
            }

/* prints true */
            if( #n >= 0 ) {
               print "true";
            } else {
               prerr "false";
            }

/* prints true */
            if( #n <= 0 ) {
               print "true";
            } else {
               prerr "false";
            }


/* prints true */
            if( #n <= ( 3 + #n) ) {
               print "true";
            } else {
               prerr "false";
            }

            set $string, "";

/* prints false */
            if( $string ) {
               print "true";
            } else {
               prerr "false";
            }

            set $string, "e";

/* prints true */
            if( $string > $string2 ) {
               print "true";
            } else {
               prerr "false";
            }


}
```

8.3.3 brainf_ck.gscc

```
[head] line {
        print "#include <stdio.h>\n";
        print "#include <stdlib.h>\n";
        print "#define INC ++*p;\n";
        print "#define DEC --*p;\n";
        print "#define SRT p++;\n";
        print "#define SLT p--;\n";
        print "#define LB while(*p){\n";
        print "#define GC *p=getchar();\n";
```

```
        print "char *p; \n";
        print "void pr(){ putchar(*p);} \n";
        print "void main(){  p = malloc(32768); char *q = p; int i; for(
i=0; i<32768; i++){*q=0; ++q;} \n";
        set @match, " ";
}
[end] line {
        set @match, "} ";
}

[>] global {
        set @match, " SRT ";
}

[<] global {
        set @match, " SLT ";
}

[\+] global {
        set @match, " INC ";
}

[-] global {
        set @match, " DEC ";
}

[\.] global {
        set @match, " pr(); ";
}

[\[] global {
        set @match, " LB ";
}
[\]] global {
        set @match, " } " ;
}
[,] global {
        set @match, " GC " ;
}
[.] line {
        //insert @line, @line.length-2, "  ";
        print @line;

}
```

8.4 Main program

8.4.1 gscc.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;
import interpreter.*;
import gsccAntlr.*;
import antlr.RecognitionException;
import antlr.TokenStreamException;
```

```
/**
 * @author ericgar
 *
 */
public class gscc {

	static boolean verbose = false;
	private static String[] gargs;
	private static int fileCount = 1;
	/**
	 * @param args
	 */
	public static void main(String[] args) {

		/*
		 * We accept arbitrary numbers of text files
		 * Absence of text file implies stdin.
		 */
		if(args.length < 1){
			System.out.println("Usage: gscc.jar <program file>
[<text file> ...]");
			return;
		}

		/* open the file that contains the gscc program definition
*/
		String programfile = args[0];
		FileInputStream program;
		try {
			program = new FileInputStream(programfile);
		}
		catch (FileNotFoundException e){
			System.err.print("Program File Not Found. " + e);
			return;
		}
		catch (Exception e){
			System.err.print("Unknown I/O Error. " + e);
			return;
		}

		ccgsLexer lexer = new ccgsLexer(program);
		ccgsParser parser = new ccgsParser(lexer);
		try{
			parser.program();
		} catch (RecognitionException e) {
			System.err.print("Recognition Error. " + e);
			return;

		} catch (TokenStreamException e) {
			System.err.print("Token Stream Error. " + e);
			return;

		}


		CommonAST tree = (CommonAST) parser.getAST();
```

```
                ccgsWalker walker = new ccgsWalker();
                Interpreter interpreter = null;
                try {
                        interpreter = walker.program(tree);
                } catch (RecognitionException e1) {
                        System.err.println("There was an error at line: " +
e1.line);
                        System.err.println("\t" + e1.getMessage());
                        System.exit(-1);
                } catch (GsccCodeException e1) {
                        if(e1.getOffendingNode() != null)
                                System.err.println("There was an error at
line " + e1.getOffendingNode().getLine()+ "\n\t" + e1.toString());
                        else{
                                System.err.println("There was an error: " +
e1.getMessage());
                                //e1.printStackTrace();
                        }
                        System.exit(-1);
                }
                 if ( lexer.nr_error > 0 || parser.nr_error > 0 ) {
            System.err.println( "Cannot Execute" );
            return;
        }
                if(verbose){
                        System.out.println( tree.toStringList() );
                        ASTFrame frame = new ASTFrame("The tree constructed
for ", tree);
                        frame.setVisible(true);
                }

                //store filenames passed as command line parameters
                gargs = args;
                while(hasNextInputStream()){
                        try {
                                interpreter.runInput(getNextInputStream(),
tree);
                        }
                        catch (GsccCodeException e1) {
                                if(e1.getOffendingNode() != null)
                                        System.err.println("There was an
error at line " + e1.getOffendingNode().getLine()+ "\n\t" + e1.toString());
                                else{
                                        System.err.println("There was an
error: " + e1.getMessage());
                                        //e1.printStackTrace();
                                }
                                System.exit(-1);
                        }
                        catch(Exception e){
                                System.err.print("Interpreter error. " +
e);
                                //e.printStackTrace();
                                return;
                        }
                }
```

```
                return;
        }


        private static boolean hasNextInputStream(){

                if(fileCount == 0){
                        return false;
                }

                return true;
        }

        private static BufferedReader getNextInputStream(){

                if(fileCount == 0){
                        return null;
                }

                if(fileCount == 1 && gargs.length == 1){
                        fileCount = 0;
                        return new BufferedReader(new
InputStreamReader(System.in));
                }

                String filename = gargs[fileCount];

                if(gargs.length > fileCount + 1){
                        fileCount++;
                }
                else {
                        fileCount = 0;
                }

                BufferedReader text;
                try {
                        text = new BufferedReader(new InputStreamReader(new
FileInputStream(filename)));
                }
                catch (FileNotFoundException e){
                        System.err.println("Target text file " + filename +
" not found.");
                        return null;
                }
                catch (Exception e){
                        System.err.println("Unknown error while opening " +
filename + ". " + e);
                        return null;
                }

                return text;

        }

}
```

8.4.2 JavaTest.java

```
import java.util.regex.*;
import java.util.*;
import interpreter.*;
import types.*;
import java.io.*;
/**
 * This file is for my own personal testing
 * @author Casey Callendrello
 *
 */

public class JavaTest {

	public JavaTest() {
		super();
		// TODO Auto-generated constructor stub
	}

	public static void main(String[] args) throws Exception{
		BufferedReader in = new BufferedReader (new
FileReader("Mary.txt"));
		LineReader l = new LineReader(in);
		String line;
		while( (line = l.readLine()) != null){
			System.out.print(line);
		}

	}

	private static void p(String val){
		System.out.println(val);
	}

}
```

8.4.3 ParseTester.java

```
import interpreter.Utils;

import java.io.*;
import antlr.*;
import antlr.debug.misc.*;
import gsccAntlr.*;

public class ParseTester {

	/**
	 * @param args
	 */
	public static void main(String[] args) throws Exception {
```

```
                String  filename = "simple.gscc";
                FileInputStream fis = new FileInputStream(filename);

                ccgsLexer lex = new ccgsLexer(fis);
                ccgsParser parse = new ccgsParser(lex);

                parse.program();

                CommonAST tree = (CommonAST)parse.getAST();


                boolean run = true;

                if(!run){
                ASTFrame frame = new ASTFrame("AST JTree Example", tree);
                frame.setVisible(true);
                }


                if( run ){
                ccgsWalker walker = new ccgsWalker();
                interpreter.Interpreter interp = walker.program(tree);
                Utils.debug("Initial walk complete");

                FileReader fr = new FileReader("test.in");
                interp.runInput(new BufferedReader(fr), null);
                }
        }

}
```

8.5 gsccAntlr

8.5.1 ccgsGrammar.g

```
/*
        ccgsGrammar.g : Lexer and parser

        @author Casey Callendrello - cdc2107@columbialedu
        @author Eric Garrido - ekg2002@columbia.edu

*/
header {
        package gsccAntlr;
}

/**
        Lexes the input program file according to the gscc language
specification.
        @author Casey Callendrello - cdc2107@columbialedu
        @author Eric Garrido - ekg2002@columbia.edu

*/
```

```
class ccgsLexer extends Lexer;

options{
    k = 4;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = ccgs;

}

{
    public int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}


protected
ALPHA   : 'a'..'z' | 'A'..'Z' | '_' ;

protected
LOWERCASE : 'a' .. 'z';
protected
DIGIT   :   '0'..'9';

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

NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
           { $setType(Token.SKIP); newline(); }
        ;
/**
        This is a modification of the comment rule shown in class.
*/
COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NL)
                    | ~( '\n' | '\r' )
                )* "*/"
           | "//" (~( '\n' | '\r' ))* (NL)
          )                       { $setType(Token.SKIP); }
        ;

/**
        Regular expressions (which begin regex blocks) are surrounded by
brackets
*/
REGEX : LBRACKET!
                 (
                 ~(']') | ('\\'! ']')
                 )*
```

```
                RBRACKET!
        ;

/* This section contains our keywords */
FUNC    : "func";
SET             : "set";
REPLACE : "replace";
RETURN  : "return";
LENGTH  : "length";
LINE            : "line";
GLOBAL  : "global";
DELETE  : "delete";
INSERT  : "insert";
PRINT   : "print";
START   : "start";
END             : "end";
IF              : "if";
WHILE   : "while";
ELSE            : "else";
PRERR   : "prerr";
BREAK   : "break";
MATCH   : "match";


/* This section contains our symbols */
LBRACKET: '[';
RBRACKET: ']';
SLASH   : '/';
LPAREN  : '(';
RPAREN  : ')';
LBRACE  : '{';
RBRACE  : '}';
SEMI    : ';';
COMMA   : ',';
DOT     : '.';
STAR    : '*';
PLUS    : '+';
MINUS   : '-';
BSLASH  : '\\';
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
LOCPFX  : '@';
NUMPFX  : '#';
STRPFX  : '$';
BANG    : '!';
AND             : "&&";
OR              : "||";


NUMBER : (DIGIT)+;

/** This rule for strings was taken from http://www.antlr.org/doc/lexer.html
*/
```

```
STRING: '"'! ( ESCAPE | ~'"' )* '"'! ;

/**
        We must properly escape characters that occur within the file.
*/
protected
ESCAPE
    :    '\\'
         ( 'n' { $setText("\n"); }
         | 'r' { $setText("\r"); }
         | 't' { $setText("\t"); }
         | '"' { $setText("\""); }
         | '\\' { $setText("\\"); }
         )
    ;

/**
        number identifiers must begin with a '#' and are only integers
*/
NUM_ID
options {
    testLiterals = true;
}
    : NUMPFX (ALPHA | DIGIT )+ ;

/**
        string identifiers must begin with a '$' and are only integers
*/
STR_ID
options {
    testLiterals = true;
}
    : STRPFX (ALPHA | DIGIT )+ ;

LOC_ID
options {
    testLiterals = true;
}
    //: LOCPFX (ALPHA | DIGIT )+ ;
    :LOCPFX (MATCH | LINE);


/**
        Parses the lexical output of the input program file according to the
gscc language specification.
        @author Casey Callendrello - cdc2107@columbialedu
        @author Eric Garrido - ekg2002@columbia.edu

*/
class ccgsParser extends Parser;

options{
    k=4;
    buildAST = true;
    exportVocab = ccgs;
}
```

```
tokens {
    STATEMENT;
    FUNC_CALL;
    FUNC_DEF;
    VAR_LIST;
    EXPRESSION;
    EXPR_LIST;
    COMMAND;
    REGEXP_BLOCK;
    UMINUS;
    ID;
    HEAD;
    PARAM_LIST;
    NUM_LIT;
}

{
    public int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}


/**
        a program consists of a series of function declarations, set
commands to initialize
        global variables, and regular expression blocks that specify
operations to perform
        when text that matches the given regular expression is processed.
*/
program
    : (regexp_block | function_def | set_cmd)* EOF! { #program =
#([HEAD,"HEAD"], program); }
    ;

/**
        since there always is a type-specifier preceeding an identifier, we
need not
        store what type of variable the identifier belongs to.
*/
id
    : (STR_ID | NUM_ID | LOC_ID){#id = #([ID,"ID"], id) ;};

/**
        several commands are restricted to types that contain only text and
cannot
        operate on numbers.
*/
string_only_id
        : (STR_ID | LOC_ID){#string_only_id = #([ID, "ID"],
string_only_id);}
```

```
        ;

// ***** Expression rules *****
 /**
        top level of the expression tree. all expressions can be used as
        a boolean term for conditional evaluation
 */
expression
        : bool_expr
        ;

// ***** Boolean expressions ******
/**
        we set up boolean expressions to preserve order of evaluation by
using
        four rules.
*/
bool_expr
        : bool_term ( OR^ bool_term )*
        ;
bool_term
        : bool_factor ( AND^ bool_factor)*
        ;
bool_factor
        : (BANG^)? bool_test
        ;
bool_test
        : num_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^ ) num_expr)*
        ;


num_expr
    : num_term ( (PLUS^ | MINUS^) num_term )*
    ;

num_term
    : num_factor ( (STAR^ | SLASH^) num_factor )*
    ;

num_factor
    : MINUS! num_value  {#num_factor = #([UMINUS,"UMINUS"], num_factor); }
    | num_value
    ;

num_value
    : id
    | func_call
    | NUMBER { #num_value = #([NUM_LIT,"NUM_LIT"], num_value); }
    | STRING
    | attribute
    | LPAREN! expression RPAREN!
    ;


// ***** Statement definitions *****
```

```
statement

        : sys_cmd
        | while_stmt
        | if_stmt
        | break_stmt
        | return_stmt
        | func_call_stmt
        | LBRACE! (statement)* RBRACE!
          {#statement = #([STATEMENT,"STATEMENT"], statement); }
        ;

break_stmt
        : BREAK^ SEMI!
        ;
return_stmt
        : RETURN^ (expression)? SEMI!
        ;
func_call_stmt
        :func_call SEMI!
        ;

func_call
        : (id) LPAREN! expr_list RPAREN!
          {#func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
        ;
/*
str_func_call
        : (STR_ID) LPAREN! expr_list RPAREN!
          {#str_func_call = #([FUNC_CALL,"FUNC_CALL"], str_func_call); }
        ;
*/
expr_list
        :  (expression (COMMA! expression)*
        | /* nothing */ )
          {#expr_list = #([EXPR_LIST, "EXPR_LIST"], expr_list); }
        ;
while_stmt
        : WHILE^ LPAREN! expression RPAREN! statement
        ;
if_stmt
        : IF^ LPAREN! expression RPAREN! statement
          (options {greedy = true;}: ELSE! statement )?
        ;

sys_cmd
        : ( set_cmd
        | setchar_cmd
        | delete_cmd
        | insert_cmd
        | replace_cmd
        | print_cmd
        | prerr_cmd)
          //{#sys_cmd = #([COMMAND, "COMMAND"], sys_cmd); }
        ;

set_cmd
```

```
                : LOC_ID DOT^ attr_name
                ;

attr_name
                : LINE
                | START
                | END
                | LENGTH
                ;

/*
location_line
        : LOC_ID DOT^ LINE
        //{} // java code goes here
        ;

location_start
        : LOC_ID DOT^ START
        //{} // java code goes here
        ;

location_end
        : LOC_ID DOT^ END
        {} // java code goes here
        ;

location_length
        : LOC_ID DOT^ LENGTH
        {} // java code goes here
        ;
*/
```

8.5.2 ccgsWalker.g

```
/*
ccgsWalker.g
The tree walker.


*/
header {
        package gsccAntlr;
}
{
        import java.io.*;
        import java.util.*;
        import types.*;
        import interpreter.*;
}

class ccgsWalker extends TreeParser;
options{
        importVocab = ccgs;
}
{
        static NumType zero = new NumType(0);
```

```
        static Interpreter interp;
        static int id = 0;
        public int myid = id++;
}


statement returns [Object o] throws GsccCodeException
{
        o = null;
        DataType a = null, b = null;
        //LocationType l;
        ExpressionList e;
        antlr.collections.AST f, g, h;
        String s;
        ExpressionList explist = new ExpressionList();
        if(! interp.funcCanProceed())
                return null;
}
        :#(SET s=id a=expr){
                //f=#ID;
                explist.append(a);
                interp.runCommand( #SET, "set",s,explist);
        }
        | #(DELETE s=id (a=expr (b=expr)?)?){
                //f=#ID;
                explist.append(a);
                explist.append(b);
                interp.runCommand(#DELETE, "delete",s, explist);
        }
        | #(INSERT s=id a=expr b=expr){
                //f=#ID;
                explist.append(a);
                explist.append(b);
                interp.runCommand(#INSERT, "insert",s, explist);
        }
        | #(REPLACE s=id a=expr b=expr){
                //f=#ID;
                explist.append(a);
                explist.append(b);
                interp.runCommand(#REPLACE, "replace",s, explist);

        }
        | #(PRINT a=expr){
                explist.append(a);
                interp.runCommand(#PRINT, "print","",explist);
        }
        | #(PRERR a=expr){
                explist.append(a);
                interp.runCommand(#PRERR, "prerr","",explist);
        }
        | #(WHILE condition:. whilebody:.){
                int id = interp.whileInit(#condition);
                while(interp.whileCanProceed(id) &&
expr(#condition).getNumVal() != 0 ){
                        statement(#whilebody);
                }
                interp.whileEnd(id);
```

```
        }
        | #(IF a=expr thenp:. (elsep:.)?) {
                if(a.getBoolVal()){
                        interp.startBlock();
                        statement(#thenp);
                        interp.endBlock();
                }else if ( null != elsep){
                        interp.startBlock();
                        statement(#elsep);
                        interp.endBlock();
                }
        }
        | #(RETURN (a=expr )?) {
                interp.setReturn(a);
        }
        | #(BREAK (.)?) { // the child part makes it compile. it will never
have a child nor will it affect runtime.
                interp.breakWhile(0);
        }
        | #(STATEMENT (state:. { statement(#state);})* )
        | #(FUNC_CALL s=id e=exlist){
                interp.callFunction(s,e);
        }
;

expr returns [DataType r] throws GsccCodeException
{
        DataType a, b;
        //LocationType l;
        ExpressionList e;
        antlr.collections.AST f, g, h;
        String s;
        r=zero;
}
        : #(OR a=expr right_or:.) {
                r = a.or( expr(#right_or) );
        }

        | #(AND a=expr right_and:.) {
                        r = a.and( expr(#right_and) );
                }

        | #(BANG a=expr) {
                        r = a.inv();
                }
        | #(GE a=expr b=expr) {
                        r=a.ge(b);
                }
        | #(LE a=expr b=expr) {
                        r=a.le(b);
                }
        | #(GT a=expr b=expr) {
                        r=a.gt(b);
                }
        | #(LT a=expr b=expr) {
                        r=a.lt(b);
                }
```

```
        | #(EQ a=expr b=expr) {
                        r=a.eq(b);
                }
        | #(NEQ a=expr b=expr) {
                        r=a.neq(b);
                }
        | #(PLUS a=expr b=expr) {
                        r=a.plus(b);
                }
        | #(MINUS a=expr b=expr) {
                        r=a.minus(b);
                }
        | #(STAR a=expr b=expr) {
                        r=a.times(b);
                }
        | #(SLASH a=expr b=expr) {
                        r=a.divide(b);
                }
        | #(UMINUS a=expr){
                        r=a.uneg();
                }
        | #(FUNC_CALL s=id e=exlist){
                r = interp.callFunction(s,e);
        }
    | #(ID name3:.){
                r = interp.getVariable(#name3.getText());
    }
    | #(DOT LOC_ID name4:.) {
                r = interp.getAttrib(#LOC_ID.getText(), name4.getText());
    }
        | #(STRING (name5:.)?){
                r = new StringType(#STRING.getText());
        }
        | #(NUM_LIT NUMBER) {
                r = new NumType(Integer.parseInt(#NUMBER.getText()));
        }
        ;


exlist returns [ExpressionList r] throws GsccCodeException
{
        r = new ExpressionList();
        DataType a, b;
}
        : #(EXPR_LIST  (a=expr { r.append(a); })*
                )
        ;

parameterlist returns [ParamList r]
{
        antlr.collections.AST a;
        r = new ParamList();
}
: #( PARAM_LIST (ID { r.append(#ID.getFirstChild().getText()); } )* )
;

program returns [Interpreter r] throws GsccCodeException
```

```
{
        antlr.collections.AST a, b, c;
        ParamList l;
        DataType d;
        String s;
        ExpressionList explist = new ExpressionList();
        if(interp == null){
                interp = new InterpreterImpl(this);
        }
        r = interp;
}
        : (#( REGEXP_BLOCK REGEX LINE STATEMENT))=>#( REGEXP_BLOCK REGEX
LINE STATEMENT){
                        a = #REGEX;
                        c = #STATEMENT;
                        r.registerRegexBlock(a.getText(),"line",c);
                }
        | #( REGEXP_BLOCK REGEX GLOBAL bodyrg:.){
                        a = #REGEX;
                        r.registerRegexBlock(a.getText(),"global",#bodyrg);
                }
        | #( FUNC s=id l=parameterlist bodyf:.){
                        //a = #ID;
                        r.registerFunction(s,l,#bodyf);
                }
        | #(SET s=id d=expr){
                //f=#ID;
                explist.append(d);
                interp.runCommand( #SET, "set",s,explist);
        }
        | #( HEAD (progline:. {program(#progline);})*)
;
id returns [String s]{
        s = "";
}

        : #(ID name:.){
                s = name.getText();
                }
        ;
```

8.6 Types

8.6.1 ExpressionList.java

```
package types;
import java.util.Vector;

/**
 *
 * @author Casey Callendrello
 *
 */
```

```
public class ExpressionList {
        private Vector<DataType> arglist;
        public ExpressionList(){
                arglist = new Vector<DataType>();
        }

        public void append(DataType val){
                arglist.add(val);
        }
        public Vector<DataType> getList(){
                return arglist;
        }
        public DataType get(int i){
                return arglist.get(i);
        }
        public int size(){
                return arglist.size();
        }

}
```

8.6.2 Location.java

```
package types;
/**
 *
 * @author Casey Callendrello
 *
 */
public interface Location {
        public int getStart();
        public int getLength();
        public int getEnd();
        public int getLine();
}
```

8.6.3 NumType.java

```
/*
 * NumType.java
 * Represents numbers and their operations within CCGS.
 */

package types;
import interpreter.GsccCodeException;

/**
 *
 * @author Casey Callendrello
 *
 */
```

```
public class NumType extends DataType {
        private int value;

        public NumType(){
                value = 0;
        }

        public NumType( int value){
                this.value = value;
        }

        public NumType( NumType type){
                this.value = type.value;
        }
        public NumType( String value){
                this.value = Integer.parseInt(value);
        }

        public NumType clone(){
                return new NumType(value);
        }

        public void set(DataType t) throws GsccCodeException{
                if(t == null){
                        throw new GsccCodeException("Attempt to set to null
value");
                }
                if(!t.isNum()){
                        new GsccCodeException("Attempt to set number to
string");
                }
                this.value = t.getNumVal();
        }

        /*
         * Boolean functions return either 1 or 0
         */
        public DataType or(DataType type){
                return new NumType((getBoolVal() || type.getBoolVal()) ? 1
: 0);
        }

        public DataType and(DataType type){
                return new NumType((getBoolVal() && type.getBoolVal()) ? 1
: 0);
        }

        public DataType inv(){
                return new NumType((getBoolVal() ) ? 0 : 1);
        }

        public DataType ge(DataType type){
                if(type.isString()){
                        return ((StringType)type).le(this);
                }
                else if(type.isNum()){
```

```
                        return new NumType( (this.value >=
type.getNumVal()) ? 1 : 0);
                }
                else{
                        return new NumType(-1);//never reached
                }
        }
        public DataType le(DataType type){
                if(type.isString()){
                        return ((StringType)type).ge(this);
                }
                else if(type.isNum()){
                        return new NumType( (this.value <=
type.getNumVal()) ? 1 : 0);
                }
                else{
                        return new NumType(-1);//never reached
                }
        }
        public DataType gt(DataType type){
                if(type.isString()){
                        return ((StringType)type).lt(this);
                }
                else if(type.isNum()){
                        return new NumType( (this.value > type.getNumVal())
? 1 : 0);
                }
                else{
                        return new NumType(-1);//never reached
                }
        }
        public DataType lt(DataType type){
                if(type.isString()){
                        return ((StringType)type).gt(this);
                }
                else if(type.isNum()){
                        return new NumType( (this.value < type.getNumVal())
? 1:0);
                }
                else{
                        return new NumType(-1);//never reached
                }
        }
        public DataType eq(DataType type){
                if(type.isString()){
                        return ((StringType)type).eq(this);
                }
                else if(type.isNum()){
                        return new NumType( (this.value ==
type.getNumVal()) ? 1: 0);
                }
                else{
                        return new NumType(-1);//never reached
                }
        }
        public DataType neq(DataType type){
                if(type.isString()){
```

```
                        return ((StringType)type).neq(this);
                }
                else if(type.isNum()){
                        return new NumType( (this.value !=
type.getNumVal()) ? 1 : 0);
                }
                else{
                        return new NumType(-1);//never reached
                }
        }


        public DataType plus(DataType type){
                if(type.isNum()){
                        return new NumType(this.value + type.getNumVal());
                }
                else{
                        return new StringType(this.value +
type.getStringVal());
                }
        }
        public DataType minus(DataType type) throws GsccCodeException{
                if(!type.isNum()){
                        throw new GsccCodeException("Trying to subtract
something that isn't a number");
                }
                int val = type.getNumVal();
                return new NumType ( this.value - val);
        }
        public DataType times(DataType type) throws GsccCodeException {
                if(!type.isNum()){
                        throw new GsccCodeException("Trying to multiply
something that isn't a number");
                }
                int val = type.getNumVal();
                return new NumType ( this.value * val);
        }
        public DataType divide(DataType type)throws GsccCodeException {
                if(!type.isNum()){
                        throw new GsccCodeException("Trying to divide
something that isn't a number");
                }
                int val = type.getNumVal();
                if( val == 0){
                        throw new GsccCodeException("Try to divide by 0");
                }
                return new NumType ( this.value / val);
        }


        public DataType uneg(){
                return new NumType ( 0 - this.value);
        }


        public boolean getBoolVal(){
                return this.value != 0;
```

```
        }

        public String getStringVal(){
                return ""+value;
        }

        public int getNumVal(){
                return value;
        }

        public boolean isString(){ return false;}
        public boolean isNum(){ return true;}

}
```

8.6.4 StringType.java

```
/**
 *
 * StringType.java
 *
 * Represents strings within CCGS
 * @author Casey Callendrello
 *
 */

package types;
//import interpreter.Utils;
import interpreter.GsccCodeException;


public class StringType extends DataType {
        private String value;

        public StringType(){

        }

        public StringType(String value){
                this.value = value;
        }

        public StringType(DataType t){
                this.value = new String(t.toString());
        }

        public StringType clone(){
                return new StringType(this.value);
        }

        public void set(DataType t) throws GsccCodeException{
                if(t == null){
                        throw new GsccCodeException("Attempting to set to a
null value");
                }
```

```
                this.value = t.toString();
        }

        public StringType plus(DataType str){
                StringType ret = new StringType(this.value +
str.getStringVal());
                return ret;
        }

        public StringType concat(DataType str){
                StringType ret = new StringType(this.value +
str.getStringVal());
                return ret;
        }

        public DataType ge(DataType in){
                return new NumType( (this.value.compareTo(in.toString())>=
0) ? 1:0);
        }
        public DataType le(DataType in){
                return new NumType( (this.value.compareTo(in.toString())<=
0) ? 1:0);
        }
        public DataType gt(DataType in){
                return new NumType( (this.value.compareTo(in.toString())>
0) ? 1:0);
        }
        public DataType lt(DataType in){
                return new NumType( (this.value.compareTo(in.toString())<
0) ? 1:0);
        }
        public DataType eq(DataType in){
                interpreter.Utils.debug("compare string");
                return new NumType( (this.value.compareTo(in.toString())==
0) ? 1:0);
        }
        public DataType neq(DataType in){
                return new NumType( (this.value.compareTo(in.toString())!=
0) ? 1:0);
        }

        public void insert(int index, String value){
                if(index < 0){
                        throw new IllegalArgumentException();
                }
                if(index >= this.value.length()){
                        this.value += index;
                }
                else{
                        this.value = this.value.substring(0, index) + value
+ this.value.substring(index);
                }
        }
        public void delete(){
                this.value = "";
        }
        public void delete(int index){
```

```
                if(index < this.value.length() && index >= 0)
                        this.value = this.value.substring(0, index);
        }
        public void delete(int index, int length){
                if(index > this.value.length()){
                        return; //do nothing
                }
                if(index + length > this.value.length()){
                        this.value = this.value.substring(0, index);
                        return;
                }
                this.value = this.value.substring(0, index) +
this.value.substring(index+length);
        }

        public void replace(int index, String newValue){
                if(index >= this.value.length()){
                        this.value += newValue;
                        return;
                }
                //case 1: we are replacing within the whole string
                if(this.value.length() - index > newValue.length()){
                        this.value = this.value.substring(0, index) +
newValue + this.value.substring(index + newValue.length() );
                }
                else if(index < value.length()){
                        this.value = this.value.substring(0, index) +
newValue;
                }
                else {
                        this.value += newValue;
                }
        }


        public String getStringVal(){
                return this.value;
        }
        public String toString(){
                return this.value;
        }
        public int getNumVal(){
                /* this should never ever happen */
                System.err.println("ERROR! NUM OF STRING");
                System.exit(-1);
                return -1;
        }
        public boolean getBoolVal(){
                return this.value.length() != 0;
        }
        public DataType inv(){
                return new NumType((getBoolVal() ) ? 0 : 1);
        }

        public boolean isString(){ return true;};
        public boolean isNum(){ return false;};
```

```
}
```

8.6.5 DataType.java

```
package types;
import interpreter.GsccCodeException;

/**
 *
 * @author Casey Callendrello
 *
 */

public abstract class DataType implements Cloneable {


        public abstract String getStringVal();
        public String toString(){
                return getStringVal();
        }
        public abstract int getNumVal();
        public abstract boolean getBoolVal();

        public abstract boolean isString();
        public abstract boolean isNum();

        public DataType clone(){
                System.err.println("Error: Clone of abstract type");
                System.exit(-1);
                return null;
        }

        public DataType and(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: And not
implemented", null);

        }
        public DataType or(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: or not
implemented", null);
        }
        public DataType inv() throws GsccCodeException{
                throw new GsccCodeException("Internal Error: inv not
implemented", null);
        }
        public DataType ge(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: ge not
implemented", null);
        }
        public DataType le(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: le not
implemented", null);
        }
        public DataType gt(DataType t) throws GsccCodeException{
```

```
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType lt(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType eq(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType neq(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType plus(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: plus not
implemented", null);
        }
        public DataType minus(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType times(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType divide(DataType t) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public DataType uneg() throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }
        public void set(DataType value) throws GsccCodeException{
                throw new GsccCodeException("Internal Error: not
implemented", null);
        }

}
```

8.6.6 LineLocation.java

```
/*
 * LineLocation is a wrapper for the Locations class
 */
package types;

import interpreter.Locations;
import interpreter.GsccCodeException;
import interpreter.Utils;

/**
 *
 * @author Casey Callendrello
```

```
 *
 */

public class LineLocation extends StringType implements Location {

	private Locations source;
	public LineLocation(){
		throw new IllegalArgumentException("MatcLocation cannot be
instantiated with no source");
	}
	public LineLocation(Locations source) {
		super();
		if(source == null){
			throw new NullPointerException("source cannot be
null");
		}
		this.source = source;
	}

	@Override
	public String getStringVal() {
		return source.lineToString();
	}

	public String toString(){
		return getStringVal();
	}

	@Override
	public int getNumVal() {
		throw new IllegalArgumentException("Get num from string!");
		//return 0;
	}

	@Override
	public boolean getBoolVal() {
		if(source.lineToString().equals(""))
			return false;
		else
			return true;
	}

	@Override
	public boolean isString() {
		return true;
	}

	@Override
	public boolean isNum() {
		return false;
	}

	public StringType clone(){
		return new StringType(source.lineToString());
	}

	public DataType ge(DataType t){
```

```
                String comp = t.getStringVal();
                int ret = source.lineToString().compareTo(comp);
                return new NumType(ret >= 0? 1: 0);
        }
        public DataType le(DataType t){
                String comp = t.getStringVal();
                int ret = source.lineToString().compareTo(comp);
                return new NumType(ret <= 0? 1: 0);
        }
        public DataType gt(DataType t){
                String comp = t.getStringVal();
                int ret = source.lineToString().compareTo(comp);
                return new NumType(ret > 0? 1: 0);
        }
        public DataType lt(DataType t){
                String comp = t.getStringVal();
                int ret = source.lineToString().compareTo(comp);
                return new NumType(ret < 0? 1: 0);
        }
        public DataType eq(DataType in){
                String comp = in.getStringVal();
                Utils.debug("compare");
                //throw new IllegalArgumentException();
                //int ret = source.lineToString().compareTo(comp);
                return new NumType(source.lineToString().equals(comp)? 1:
0);
        }
        public DataType neq(DataType t){
                String comp = t.getStringVal();
                int ret = source.lineToString().compareTo(comp);
                return new NumType(ret != 0? 1: 0);
        }
        public StringType plus(DataType t){
                return new StringType( source.lineToString() +
t.getStringVal() );
        }

        public void set(DataType value) throws GsccCodeException{
                if(value == null){
                        throw new GsccCodeException("Trying to set a
variable to no value");
                }
                source.set(value.getStringVal(), 0, 0, source.getLineNo());
        }

        public void insert(int index, String value){
                source.insertLine(value,index);
        }
        public void delete(){
                source.deleteLine();
        }
        public void delete(int index){
                source.deleteLine(index);
        }
        public void delete(int index, int length){
                source.deleteLine(index, length);
        }
```

```
        public void replace(int index, String newValue){
                source.replaceLine(newValue, index);
        }
        public int getStart() {
                return 0;
        }
        public int getLength() {
                return source.lineLength();
        }
        public int getEnd() {
                int a =source.lineLength();
                if( a == 0) return a;
                return a-1;

        }
        public int getLine() {
                return source.getLineNo();
        }

}
```

8.6.7 MatchLocation.java

```
/*
 * MatchLocation is a wrapper for the match location within the DataType
framework.
 */
package types;
import interpreter.Locations;
import interpreter.GsccCodeException;

/**
 *
 * @author Casey Callendrello
 *
 */

public class MatchLocation extends StringType implements Location {

        private Locations source;
        public MatchLocation(){
                throw new IllegalArgumentException("MatcLocation cannot be
instantiated with no source");
        }
        public MatchLocation(Locations source) {
                super();
                if(source == null){
                        throw new NullPointerException("source cannot be
null");
                }
                this.source = source;
        }

        @Override
        public String getStringVal() {
```

```
        return source.matchToString();
}

public String toString(){
        return getStringVal();
}

@Override
public int getNumVal() {
        throw new IllegalArgumentException("Get num from string!");
        //return 0;
}

@Override
public boolean getBoolVal() {
        if(source.matchToString().equals(""))
                return false;
        else
                return true;
}

@Override
public boolean isString() {
        return true;
}

@Override
public boolean isNum() {
        return false;
}

public StringType clone(){
        return new StringType(source.matchToString());
}

public DataType ge(DataType t){
        String comp = t.getStringVal();
        int ret = source.matchToString().compareTo(comp);
        return new NumType(ret >= 0? 1: 0);
}
public DataType le(DataType t){
        String comp = t.getStringVal();
        int ret = source.matchToString().compareTo(comp);
        return new NumType(ret <= 0? 1: 0);
}
public DataType gt(DataType t){
        String comp = t.getStringVal();
        int ret = source.matchToString().compareTo(comp);
        return new NumType(ret > 0? 1: 0);
}
public DataType lt(DataType t){
        String comp = t.getStringVal();
        int ret = source.matchToString().compareTo(comp);
        return new NumType(ret < 0? 1: 0);
}
public DataType eq(DataType in){
        String comp = in.getStringVal();
```

```
                interpreter.Utils.debug("compare");
                //int ret = source.matchToString().compareTo(comp);
                return new NumType(source.matchToString().equals(comp)? 1:
0);
        }
        public DataType neq(DataType t){
                String comp = t.getStringVal();
                int ret = source.matchToString().compareTo(comp);
                return new NumType(ret != 0? 1: 0);
        }
        public StringType plus(DataType t){
                return new StringType(source.matchToString() +
t.getStringVal());
        }

        public void set(DataType value) throws GsccCodeException{
                if(value == null){
                        throw new GsccCodeException("Attempt to set to null
value");
                }
                source.setMatch(value.getStringVal());
        }

        public void insert(int index, String value){
                source.insertMatch(value,index);
        }
        public void delete(){
                source.deleteMatch();
        }
        public void delete(int index){
                source.deleteMatch(index);
        }
        public void delete(int index, int length){
                source.deleteMatch(index, length);
        }

        public void replace(int index, String newValue){
                source.replaceMatch(newValue, index);
        }
        public int getStart() {
                return source.matchStart();
        }
        public int getLength() {
                return source.matchLength();
        }
        public int getEnd() {
                return source.matchEnd();
        }
        public int getLine() {
                return source.getLineNo();
        }


}
```

8.6.8 ParamList.java

```
package types;
import java.util.Vector;

/**
 *
 * @author Casey Callendrello
 *
 */

public class ParamList {
        private Vector<String> plist;
        public ParamList(){
                plist = new Vector<String>();
        }

        public void append(String name){
                plist.add(name);
        }
        public Vector<String> getList(){
                return plist;
        }
        public int size(){
                return plist.size();
        }
}
```

8.7 Interpreter

8.7.1 ccgsFunction.java

```
package interpreter;
import types.DataType;
import types.ParamList;
import antlr.collections.AST;



/**
 * Represents a function block within a ccgs program.
 * @author Casey Callendrello
 *
 */
public class ccgsFunction {
        public AST node;
        public types.ParamList params;
        public boolean isNumeric;
        public boolean isInternal = false;
        public int intId = 0;


        /**
```

```
         * @param node the node representing the body of the function
         * @param list a ParamList containing the required parameters for
this function
         * @param isNumeric determines whether the function is of type
numeric or string
         */
        public ccgsFunction(AST node, ParamList list, boolean isNumeric){
                this.node = node;
                this.params = list;
                this.isNumeric = isNumeric;
        }
}
```

8.7.2 GsccCodeException.java

```
package interpreter;

import antlr.collections.AST;


/**
 * This exception class allows the interpreter to throw gscc-specific error
messages
 * at run-time while traversing the tree. It includes the offending AST node
 * for further inspect
 */
/**
 * @author Casey Callendrello
 *
 */
public class GsccCodeException extends Exception {
        private AST offendingNode;


        public GsccCodeException() {
                super();
                // TODO Auto-generated constructor stub
        }

        public GsccCodeException(String message) {
                super(message);
                // TODO Auto-generated constructor stub
        }

        public GsccCodeException(String message, Throwable cause) {
                super(message, cause);
                // TODO Auto-generated constructor stub
        }

        public GsccCodeException(Throwable cause) {
                super(cause);
                // TODO Auto-generated constructor stub
        }
```

```
        public GsccCodeException(AST offender, String message){
                super(message);
                this.offendingNode = offender;
        }

        public AST getOffendingNode(){
                return offendingNode;
        }
        public void setOffendingNode(AST node){
                this.offendingNode = node;
        }

}
```

8.7.3 InterpreterImpl.java

```
package interpreter;
import antlr.*;
import types.*;
import antlr.collections.AST;
import java.util.*;
import java.util.regex.*;
import gsccAntlr.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.concurrent.LinkedBlockingQueue;
/**
 *
 * @author Casey Callendrello
 *
 */

public class InterpreterImpl implements Interpreter {
        private static int SUBSTR = 1;
        private static int LENGTH = 2;
        private static int PARSE = 3;

        /**
         * The interpreter is a state machine.  It maintains state in the
symbol table and the function table.
         * The tree walker creates the interpreter, setting up the tables.
         * The main method then calls the interpreter, which walks the tree.
         *
         */

        /* these are the basic setup variables */
        private SymbolTable root;
        private HashMap<String, ccgsFunction> functions;
        private ccgsWalker walker;
        private LinkedList<RegexBlock> blocks;
        private BufferedReader in;
```

```
        /*variable aliases */
        private static int READY = 0;
        private static int BREAK = 1;
        private static int RETURN = 2;


        /*these are the state vars */
        private SymbolTable current;
        private int nextWhile = 0;
        private int flowState = READY;
        private DataType returnValue = null;
        Locations currentLocation;


        /**
         * Deprecated. Use the InterpreterImpl(ccgsWalker) constructor
instead.
         */
        public InterpreterImpl(){
                //this(new ccgsWalker());
                Utils.debug("do not call me");
                /*root = new SymbolTable();
                current = root;
                functions = new HashMap<String, ccgsFunction>();
                */

        }


        /**
         * Creates a new instance of the gscc interpreter which will
traverse the tree
         * using the parameter walker and perforn the operations specified.
         * @param w a properly initialized ccgsWalker
         */
        public InterpreterImpl(ccgsWalker w){
                if(w == null){System.err.println("null walker"); }
                this.walker = w;
                root = new SymbolTable();
                current = root;
                functions = new HashMap<String, ccgsFunction>();
                blocks = new LinkedList<RegexBlock>();
                registerInternalFunctions();
        }


        /* (non-Javadoc)
         * @see interpreter.Interpreter#callFunction(java.lang.String,
types.ExpressionList)
         */
        public DataType callFunction(String name, ExpressionList explist)
throws GsccCodeException, antlr.RecognitionException {
                Utils.debug("Calling function " + name);
                if(!functions.containsKey(name)){
                        throw new GsccCodeException("Call to non-existent
function " + name);
                }
```

```
                ccgsFunction f = functions.get(name);
                DataType ret;

                if(f.isInternal){
                        return callInternalFunction(f.intId, explist);
                }

                /* initialize ret to default values and set type */
                /*if(f.isNumeric)
                        ret = new NumType(0);
                else
                        ret = new StringType("");
                */

                /*
                 * Establish the names of the values and enter each value
in to
                 * the symbol table.
                 */

                SymbolTable table = new SymbolTable(root, current);

                Vector<String> argNames = f.params.getList();
                Vector<DataType> values = explist.getList();

                if(argNames.size() != values.size()){
                        throw new GsccCodeException("Incorrect number of
arguments");
                }

                //take the parameters and put their values in the symbol
table

                Enumeration<String> argNamesEnum = argNames.elements();
                Enumeration<DataType> valuesEnum = values.elements();
                //insert each var
                while(argNamesEnum.hasMoreElements()){
                        String argName = argNamesEnum.nextElement();
                        Utils.debug("put variable " + argName + " into
symbol table");
                        table.put(argName, valuesEnum.nextElement());
                }


                /*
                 * Set the global current table to this table.
                 */
                this.current = table;

                try{
                        walker.statement(f.node);
                }
                catch (RecognitionException e){
                        throw e;
                }

                /* we are done, clean up */
```

```
                this.current = current.previous;

                flowState = READY;
                ret = returnValue;
                returnValue = null;


                /*
                 * Function default values, may change.
                 */

                if(ret == null){
                        if(f.isNumeric){
                                ret = new NumType(0);
                        }
                        else{
                                ret = new StringType("");
                        }
                }
                //Check to see if return type matches function type
                else if(f.isNumeric && !ret.isNum()){
                        throw new GsccCodeException("Attempt to return
String in a numerical function");
                }
                else if(!f.isNumeric && ret.isNum()){
                        ret = new StringType(ret.getStringVal());
                }



                Utils.debug("Function " + name + " returns " +
ret.toString());
                return ret;
        }

        public void registerFunction(String name, ParamList paramlist, AST
node) throws GsccCodeException {
                Utils.debug("registering function " + name + " with " +
paramlist.size());
                if(functions.containsKey(name)){
                        throw new GsccCodeException(node, "Attempt to
register function with duplicate name");
                }
                boolean isNumeric = (name.charAt(0) == '#');
                ccgsFunction f = new ccgsFunction(node, paramlist,
isNumeric);
                functions.put(name, f);
        }

        public void runCommand(AST node, String command, String target,
ExpressionList exprlist) throws GsccCodeException {
                /*
                 * The behaivor for each command is different enough that
we switch.
                 */
```

```
                Utils.debug("running command " + command + " target " +
target + " list " + exprlist.size());

                if(command.equals("set")){
                        Utils.debug("set " + target + " as " +
exprlist.get(0));

                        if(exprlist.size() != 1){
                                throw new GsccCodeException(node, "set must
have two arguments");
                        }
                        DataType val = exprlist.get(0);

                        //check to see if the variable name and expresison
agree on type.
                        if(target.charAt(0) == '#' && val.isString()){
                                throw new GsccCodeException(node, "Trying
to assign string to number");
                        }
                        /*
                        if(target.charAt(0) == '@'){

                                return;
                        }*/

                        //insert or update the new value
                        if(current.containsKey(target)){
                                current.get(target).set(val);
                        }
                        else{
                                DataType newVal;
                                //check type and create NewVal
                                if(target.charAt(0) == '#')
                                        newVal = new
NumType(val.getNumVal());
                                else if(target.charAt(0) == '$')
                                        newVal = new
StringType(val.getStringVal());
                                else
                                        newVal = new NumType();
//unreachable code.
                                current.put(target,newVal);
                        }
                } //end set


                else if(command.equals("insert")){
                        if(exprlist.size() != 2){
                                throw new GsccCodeException(node, "insert
must have three arguments");
                        }
                        DataType location = exprlist.get(0);
                        DataType value = exprlist.get(1);

                        /*if(target.charAt(0) == '@'){

                                return;
```

```
                        }*/

                        if(target.charAt(0) == '#'){
                                throw new GsccCodeException(node, "cannot
insert in a number");
                        }

                        if(!current.containsKey(target)){
                                throw new GsccCodeException(node, "variable
" + target + " not defined");
                        }

                        DataType var = current.get(target);

                        StringType svar = (StringType)var;

                        if(!location.isNum()){
                                throw new GsccCodeException(node, "second
argument of insert must be a number");
                        }
                        svar.insert(location.getNumVal(),
value.getStringVal());
                }//end insert command


                else if (command.equals("delete")){
                        if(exprlist.size() != 2){
                                throw new GsccCodeException(node, "delete
must have between 1 and 3 arguments");
                        }

                        /*
                        if(target.charAt(0) == '@'){

                                return;
                        }
                        */

                        if(!current.containsKey(target)){
                                throw new GsccCodeException(node, "variable
" + target + " not defined");
                        }
                        DataType var = current.get(target);

                        if(!var.isString()){
                                throw new GsccCodeException(node, "variable
" + target + " is not a string");
                        }
                        StringType svar = (StringType) var;

                        if(exprlist.size() == 0){
                                svar.delete();
                        }
                        else if(exprlist.size() == 1){
                                DataType location = exprlist.get(0);
                                if(!location.isNum())
```

```
                                        throw new GsccCodeException(node,
"second argument of delete must be a number");
                                svar.delete(location.getNumVal());
                        }
                        else if(exprlist.size() == 2){
                                DataType location = exprlist.get(0);
                                if(!location.isNum())
                                        throw new GsccCodeException(node,
"second argument of delete must be a number");
                                DataType length = exprlist.get(1);
                                if(!length.isNum())
                                        throw new GsccCodeException(node,
"third argument of delete must be a number");

                                svar.delete(location.getNumVal(),
length.getNumVal());
                        }

                }//end delete
                else if (command.equals("replace")){

                        /*
                        if(target.charAt(0) == '@'){
                                return;
                        }
                        */

                        if(!current.containsKey(target)){
                                throw new GsccCodeException(node, "variable
" + target + " not defined");
                        }
                        DataType var = current.get(target);

                        if(!var.isString()){
                                throw new GsccCodeException(node, "variable
" + target + " is not a string");
                        }

                        StringType svar = (StringType)var;
                        if(exprlist.size() != 2){
                                throw new GsccCodeException(node, "replace
must have three arguments");
                        }

                        DataType location = exprlist.get(0);
                        if(!location.isNum())
                                throw new GsccCodeException(node, "second
argument of replace must be a number");
                        DataType value = exprlist.get(1);


        svar.replace(location.getNumVal(),value.getStringVal());
                }//end replace

                else if(command.equals("print")){
                        if(exprlist.size() != 1){
```

```
                                throw new GsccCodeException(node, "print
must have one argument");
                        }
                        System.out.print(exprlist.get(0).toString());
                }
                else if(command.equals("prerr")){
                        if(exprlist.size() != 1){
                                throw new GsccCodeException(node, "prerr
must have one argument");
                        }
                        System.err.print(exprlist.get(0).toString());
                }

                //do NOT put code here
                //any of the if blocks may return at any time.

        }//end function

        public DataType getVariable(String name) throws GsccCodeException {
                Utils.debug("retrieve var " + name);
                if(!current.containsKey(name))
                        throw new GsccCodeException("variable " + name + "
does not exist");
                if(name.charAt(0) == '@'){
                        return new
StringType(current.get(name).getStringVal());
                }
                return current.get(name).clone();
        }

        public DataType getAttrib(String name, String attrName) throws
GsccCodeException {
                DataType v = current.get(name);
                if(name.charAt(0) != '@'){
                        throw new GsccCodeException("Non-location variables
do not have attributes");
                }
                Location l = (Location) v;
                DataType ret;
                if(attrName.equals("line")){
                        ret = new NumType(l.getLine());
                }
                else if(attrName.equals( "start")){
                        ret = new NumType(l.getStart());
                }
                else if(attrName.equals( "end")){
                        ret = new NumType(l.getEnd());
                }
                else if(attrName.equals( "length")){
                        ret = new NumType(l.getLength());
                }
                else{
                        throw new GsccCodeException("That is not a
recognized attribute");
                }

                return ret;
```

```
        }

        public void registerRegexBlock(String regex, String type, AST node)
{
                Utils.debug("Registering " + regex + " "+ type + " " +
node.getText());
                boolean isGlobal = (type == "global");
                RegexBlock b = new RegexBlock(regex, isGlobal, node);
                blocks.add(b);
        }

        public void runInput(java.io.BufferedReader in, AST program) throws
Exception{
                this.in = in;
                LineReader reader = new LineReader(in);
                if(blocks.size() == 0){ System.err.println("no regex blocks
found"); return;}

                //this sets up function definitions and regexp blocks
                //walker.program(program);

                //we need to know the newline since readLine() doesn't
return them
                String newline = System.getProperty("line.separator");

                //line number
                int lineNumber = 1;

                //The queue is added to when a modification is made that
adds a new line.
                Queue<String> lines = new LinkedBlockingQueue<String>();
                String line =reader.readLine();// in.readLine();
                Utils.debug("read line " + line);
                if(line == null) return;
                lines.add(line);
                currentLocation = new Locations("", 0, 0, lineNumber);


                LineLocation atLine = new LineLocation(currentLocation);
                MatchLocation atMatch = new MatchLocation(currentLocation);


                Utils.debug("Start main loop");
                while(!lines.isEmpty()){ //this is the program loop
                        line = lines.remove(); //take the next line
                        ListIterator<RegexBlock> iter =
blocks.listIterator(0);

                        while(iter.hasNext()){ //for each regex block
                                RegexBlock curr = iter.next();
                                Utils.debug("running on line " + line + "
with block " + curr.pattern.toString());
                                Matcher m = curr.pattern.matcher(line);

                                if(!curr.isGlobal){ //we are in line mode
                                        if(m.find()){ //if we have a match
```

```
                                                currentLocation.set(line,
m.start(), m.end(), lineNumber);
                                                current = new
SymbolTable(root, null); //new symbol table
                                                current.put("@line",
atLine);
                                                current.put("@match",
atMatch);
        walker.statement(curr.node);//run it!
                                                current = root; //discard
this symbol table
                                                line =
currentLocation.lineToString();
                                        }
                                }
                                else if(curr.isGlobal){ //we are in global
mode
                                        int lastMatch = 0;
                                        //m.reset(line);
                                        try{
                                                while(m.find(lastMatch)){
//while there are matches
                                                        lastMatch =
m.end(); //get the last char we've searched to
        currentLocation.set(line, m.start(), m.end(), lineNumber); //set up
locations
                                                        current = new
SymbolTable(root, null); // new symbol table
        current.put("@line", atLine);
        current.put("@match", atMatch);
        walker.statement(curr.node);//run it!
                                                        current = root;
//discard symbol table.
                                                        line =
currentLocation.lineToString(); //get the new line
                                                        m.reset(line);
//set the matcher to use this new line
                                                        if(lastMatch >=
line.length())
                                                                break;
                                                }
                                        }
                                        catch(IndexOutOfBoundsException e){
                                                //If the line is modified
and we try to search too far.
                                                //do nothing.
                                        }
                                }
```

```
                        }
                        if(lines.isEmpty()){
                                line = reader.readLine();
                                lineNumber++;
                                if(line != null)
                                        lines.add(line);
                        }
                }

        }

        public void startBlock(){
                SymbolTable t = new SymbolTable(current, current);
                current = t;
        }

        public void endBlock(){
                current = current.previous;
        }
        public int whileInit(AST node){
                startBlock();
                return nextWhile++;
        }
        public boolean whileCanProceed(int id) {
                return flowState == READY;
        }

        public void whileEnd(int id){
                nextWhile--;
                endBlock();

                /* if flowstate = return, we keep it so functions will
return */
                if(flowState == BREAK)
                        flowState = READY;

        }

        public void breakWhile(int id){
                flowState = BREAK;
        }

        public boolean funcCanProceed(){
                if(flowState != READY){
                        return false;
                }
                return true;
        }

        public void setReturn(DataType value){
                Utils.debug("Caught return: " + value.toString());
                returnValue = value.clone();
                flowState = RETURN;
        }

        private void registerInternalFunctions(){
```

```
                ParamList p = new ParamList();
                p.append("$variable");
                p.append("#start");
                p.append("#end");
                ccgsFunction f = new ccgsFunction(null, p, false);
                f.intId=SUBSTR;
                f.isInternal = true;
                functions.put("$substr", f);


                p = new ParamList();
                p.append("$variable");
                f = new ccgsFunction(null, p, false);
                f.intId=LENGTH;
                f.isInternal = true;
                functions.put("#length", f);

                p = new ParamList();
                p.append("$variable");
                f = new ccgsFunction(null, p, false);
                f.intId=PARSE;
                f.isInternal = true;
                functions.put("#parse", f);


        }
        private DataType callInternalFunction(int id, ExpressionList e)
throws GsccCodeException{
                DataType ret = null;
                if(id == SUBSTR){
                        if(e.size() !=3) throw new
GsccCodeException("Substring must have 3 parameters");

                        String val = e.get(0).getStringVal();
                        int start = e.get(1).getNumVal();

                        int end = e.get(2).getNumVal() + start;
                        if(start >= val.length()){
                                return new StringType("");
                        }
                        if(end >= val.length()){
                                ret = new StringType(val.substring(start));
                        }
                        else
                                ret = new StringType(val.substring(start,
end));
                }
                else if(id == LENGTH){
                        if(e.size() != 1) throw new
GsccCodeException("Length must have 1 parameter");
                        String val = e.get(0).getStringVal();
                        ret = new NumType(val.length());
                }
                else if(id == PARSE){
                        if(e.size() != 1) throw new
GsccCodeException("Length must have 1 parameter");
                        String val = e.get(0).getStringVal();
```

```
                        try{
                                ret = new NumType(Integer.parseInt(val));
                        }
                        catch(NumberFormatException ex){
                                ret = new NumType(0);
                        }
                }

                return ret;

        }
}

class RegexBlock {
        //public String regex;
        public Pattern pattern;
        public boolean isGlobal;
        public AST node;

        public RegexBlock(String regex, boolean isGlobal, AST node){
                //this.regex = regex;
                try{
                        this.pattern = Pattern.compile(regex,
Pattern.DOTALL);
                }
                catch(PatternSyntaxException e){
                        System.err.println("Incorrect pattern " + regex);
                }
                this.isGlobal = isGlobal;
                this.node = node;
        }

}
```

8.7.4 Locations.java

```
package interpreter;

//import interpreter.InterpreterImpl.Locations.ListNode;

public class Locations {
        /**
         * Locations is a class that manages the linked list of characters.
         * This linked list allows for easy insertion and deletions.
         * @author Casey Callendrello
         *
         */
        class ListNode {
                public char value;

                public ListNode next;

                public ListNode prev;

                ListNode(char v) {
```

```
                        this.value = v;
                }
        }

        ListNode head, tail, matchStart, matchEnd;
        int lineNo;

        //boolean matchIsNull = false;

        /**
         * Set initializes the locations values.  Begin and end are
numerical indexes of the match value.
         * end = begin + length
         * the matchEnd pointer is one past the end.
         */
        public Locations(String value, int begin, int end, int lineNo) {
                this.lineNo = lineNo;
                Utils.debug("set with val " + value + " at " + begin + "
and " + end);
                head = new ListNode('!');
                tail = new ListNode('!');
                matchStart = null;
                matchEnd = null;
                //end++;

                ListNode curr = head;
                for (int i = 0; i < value.length(); i++) {
                        ListNode n = new ListNode(value.charAt(i));
                        curr.next = n;
                        n.prev = curr;
                        n.next = tail;
                        tail.prev = n;
                        curr = n;
                        if (i == begin)
                                matchStart = n;
                        if (i == end)
                                matchEnd = n;
                }
        }

        public void set(String value, int begin, int end, int lineNo) {
                head = new ListNode('!');
                tail = new ListNode('!');
                matchStart = null;
                matchEnd = null;
                this.lineNo = lineNo;
                //end++;

                ListNode curr = head;
                for (int i = 0; i < value.length(); i++) {
                        ListNode n = new ListNode(value.charAt(i));
                        curr.next = n;
                        n.prev = curr;
                        n.next = tail;
                        tail.prev = n;
                        curr = n;
                        if (i == begin)
```

```
                        matchStart = n;
                if (i == end){
                        matchEnd = n;
                }
        }
        if(end == value.length()){
                matchEnd = tail;
        }
}

public String matchToString() {

        StringBuffer buff = new StringBuffer();
        ListNode curr = matchStart;
        while (curr != matchEnd && curr != tail) {
                buff.append(curr.value);
                curr = curr.next;
        }
        Utils.debug("matchToString returns " + new String(buff));
        return new String(buff);
}

public String lineToString() {
        StringBuffer buff = new StringBuffer();
        ListNode curr = head.next;
        while (curr != tail) {
                buff.append(curr.value);
                curr = curr.next;
        }
        return new String(buff);
}

public void setMatch(String value) {
        if (value.length() == 0) {
                matchStart.prev.next = matchEnd;
                matchStart = matchEnd;
        } else {
                matchStart.value = value.charAt(0);
                ListNode curr = matchStart;
                for (int i = 1; i < value.length(); i++) {
                        ListNode n = new ListNode(value.charAt(i));
                        n.prev = curr;
                        curr.next = n;
                        curr = n;
                }
                curr.next = matchEnd;
                matchEnd.prev = curr;
        }
}

public void insertMatch(String value, int index) {
        ListNode curr = matchStart;
        for (int i = 0; i < index && curr != matchEnd;) {
                curr = curr.next;
                i++;
        }
        for (int i = 0; i < value.length(); i++) {
```

```
                        ListNode n = new ListNode(value.charAt(i));
                        curr.prev.next = n;
                        n.prev = curr.prev;
                        curr.prev = n;
                        n.next = curr;
                }
        }

        public void deleteMatch() {
                matchStart.prev.next = matchEnd;
                matchStart = matchEnd;
        }

        public void deleteMatch(int index) {
                ListNode curr = matchStart;
                for (int i = 0; i < index && curr != matchEnd;) {
                        i++;
                        curr = curr.next;
                }
                curr.prev.next = matchEnd;
                matchEnd.prev = curr.prev;
        }

        public void deleteMatch(int index, int length) {

                ListNode curr = matchStart;
                for (int i = 0; i < index && curr != matchEnd;) {
                        i++;
                        curr = curr.next;
                }
                for (int i = 0; i < length && curr != matchEnd;) {
                        curr.prev.next = curr.next;
                        curr.next.prev = curr.prev;
                        i++;
                        curr = curr.next;
                }
                if(index == 0){
                        matchStart = curr;
                }
        }

        public void replaceMatch(String newValue, int index) {
                if (newValue.length() == 0)
                        return;
                ListNode curr = matchStart;
                for (int i = 0; i < index & curr != matchEnd;) {
                        curr = curr.next;
                        i++;
                }
                int i = 0;
                for (; i < newValue.length() && curr != matchEnd;) { //yes
this is a while.  go away.
                        curr.value = newValue.charAt(i);
                        curr = curr.next;
                        i++;
                }
```

```
                if (curr == matchEnd) { //this is a special case because
matchEnd is one ahead so we can't change the value
                        while (i < newValue.length()) {
                                ListNode n = new
ListNode(newValue.charAt(i));
                                curr.prev.next = n;
                                n.prev = curr.prev;
                                n.next = curr;
                                curr.prev = n;
                                i++;
                        }
                }
        }

        /*
         * Line functions
         * set
         * insert
         * delete
         * replace
         */

        public void setLine(String value) {
                ListNode curr = head.next;
                int i=0;
                boolean seenMatchStart = false, seenMatchEnd = false;
                for(; i<value.length() && curr != tail; i++){
                        curr.value=value.charAt(i);
                        if(curr == matchStart ) seenMatchStart = true;
                        if(curr == matchEnd ) seenMatchEnd = true;
                        curr = curr.next;
                }
                if(i < value.length() ){//we reached the end of the list.
curr = tail
                        while(i < value.length()){
                                ListNode n = new ListNode(value.charAt(i));
                                n.prev = curr.prev;
                                n.next = curr;
                                curr.prev.next = n;
                                curr.prev = n;
                                i++;
                        }
                }
                else if( i == value.length() && curr != tail){ //value is
shorter than current length
                        if(!seenMatchStart){//match is past the end of the
new string
                                matchStart = tail;
                                matchEnd = tail;

                        }
                        else if(seenMatchStart && !seenMatchEnd){
                                matchEnd = tail;
                        }
                        curr.prev.next = tail;
                        tail.prev = curr.prev;
                }
```

```
	}

	public void insertLine(String value, int index) {
		ListNode curr = head.next;
		for (int i = 0; i < index && curr != tail; i++) {
			curr = curr.next;
		}
		for (int i = 0; i < value.length(); i++) {
			ListNode n = new ListNode(value.charAt(i));
			curr.prev.next = n;
			n.prev = curr.prev;
			n.next = curr;
			curr.prev = n;
		}
	}

	public void deleteLine() {
		head.next = tail;
		tail.prev = head;
		matchStart = matchEnd = tail;
	}

	public void deleteLine(int index) {
		boolean seenMatchStart = false;
		boolean seenMatchEnd = false;

		ListNode curr = head.next;

		// advance curr to the start of deletion, looking for the
match pointers as we do so.
		int i = 0;
		while(i<index && curr != tail){
			if(curr == matchStart) seenMatchStart = true;
			if(curr == matchEnd) seenMatchEnd = true;
			i++;
			curr = curr.next;
		}

		if (seenMatchStart && !seenMatchEnd) {
			matchEnd = curr.prev;
		}
		/*this is undefined state: we are deleting the string with
the match */
		if (!seenMatchStart) {
			matchStart = tail;
			matchEnd = tail;
		}
		if(curr != tail){
			curr.prev.next = tail;
			tail.prev = curr.prev;
		}

	}

	public void deleteLine(int index, int length) {
		int i=0;
		ListNode curr = head;
```

```
                boolean seenMatchStart = false, seenMatchEnd = false;
                for(; i<index && curr != tail;  ){

                        if(!seenMatchStart && curr == matchStart){
                                seenMatchStart = true;
                        }
                        if(!seenMatchEnd && curr == matchEnd){
                                seenMatchEnd = true;
                        }
                        curr = curr.next;
                        i++;
                }
                if(!seenMatchStart){ //we didn't see the beginning, so we
delete, moving matchStart and matchEnd forwards as we delete
                        i = 0;
                        ListNode oldStart = curr.prev;
                        for(; i<length && curr != tail;  ){
                                if(curr == matchStart){
                                        matchStart = curr.next;
                                }
                                if(curr == matchEnd){

                                }
                                i++;
                                curr = curr.next;
                        }
                        oldStart.next = curr;
                        curr.prev = oldStart;
                }
                else if(seenMatchStart && !seenMatchEnd){
                        i = 0;
                        ListNode oldStart = curr.prev;
                        for(; i<length && curr != tail;  ){
                                if(curr == matchEnd){
                                        matchEnd = curr.next;
                                }
                                i++;
                                curr = curr.next;
                        }
                        oldStart.next = curr;
                        curr.prev = oldStart;
                }
                else if(seenMatchStart && seenMatchEnd){
                        i=0;
                        ListNode oldStart = curr.prev;
                        for(; i<length && curr != tail;  ){
                                i++;
                                curr = curr.next;
                        }
                        oldStart.next = curr;
                        curr.prev = oldStart;
                }
                //Utils.debug("matchStart char: " + matchStart.value + "
matchEnd char: " + matchEnd.value);
        }
        public void replaceLine(String source, int index){
                ListNode curr = head.next;
```

```
                for(int i=0; i<index && curr != tail; i++){
                        curr = curr.next;
                }

                for(int i=0; i<source.length() && curr != tail; i++){
                        curr.value = source.charAt(i);
                        curr = curr.next;
                }
        }
        public char tail(){
                return tail.value;
        }

        public int matchLength(){
                ListNode curr = matchStart;
                int len = 0;
                while(curr != matchEnd){
                        len++;
                        curr = curr.next;
                }
                return len;
        }

        public int lineLength(){
                ListNode curr = head.next;
                int len = 0;
                while(curr != tail){
                        len++;
                        curr = curr.next;
                }
                return len;
        }
        public int matchStart(){
                ListNode curr = head.next;
                int len = 0;
                while(curr != matchStart){
                        len++;
                        Utils.debug(curr.value+"<-");
                        curr = curr.next;
                }
                return len;
        }
        public int matchEnd(){
                ListNode curr = head.next;
                int len = 0;
                while(curr.next != matchEnd){
                        len++;
                        curr = curr.next;
                }
                return len;
        }
        public int getLineNo(){
                return lineNo;
        }
}
```

8.7.5 Utils.java

```
package interpreter;


/**
 * Utils is a static class for output debugging. Nothing to see here.
 * @author Casey Callendrello
 *
 */
public class Utils {

	public Utils() {
		super();
		// TODO Auto-generated constructor stub
	}
	static final boolean DEBUG = false;

	public static void debug(String message){
		if(DEBUG){
			System.err.println(message);
		}
	}

}
```

8.7.6 Interpreter.java

```
package interpreter;
import antlr.collections.AST;
import types.*;
import antlr.RecognitionException;

/**
 *	Interpreter is the interface for the ccgs interpreter and is called
by the tree walker
 *  as it traverses the AST tree.
 *  @author Casey Callendrello
 */
public interface Interpreter {

	/**
	 * registerFunction registers a function within
	 * @param name name of the function, including the type specifier
	 * @param paramlist parameter list of the function
	 * @param node AST node of the body of the function
	 * @throws GsccCodeException
	 */
	public void registerFunction(String name, ParamList paramlist,
antlr.collections.AST node) throws GsccCodeException;

	/**
	 * Calls a previously registered function, or an internal
	 * @param name The name of the function
	 * @param explist An ExpressionList of the passed values
```

```
         * @return Returns a DataType, optionally, of this functions return
result
         * @throws GsccCodeException Throws if there is an invalid state
(i.e. attempt to reference an unset variable)
         * @throws RecognitionException Thrown by the walker if the function
in the walker has a syntax error;
         */
        public DataType callFunction(String name, ExpressionList explist)
throws GsccCodeException, RecognitionException; //returns result

        /**
         * The walker uses this function to determine if there has been a
return called
         * @return Returns true unless setReturn has been called
         */
        public boolean funcCanProceed();

        /**
         * Runs one of the built-in commands
         * @param node A node for printing line errors if there is an errlr
         * @param name The name of the command to be called
         * @param target The target variable name, for commands like set,
intsert, etc
         * @param exprlist The list of expresions that we are operating with
         * @throws GsccCodeException
         */
        public void runCommand(AST node, String name, String target,
ExpressionList exprlist) throws GsccCodeException; //handles symbol table

        /**
         * Retrieves a variable from the present symbol table or any of its
parents
         * @param name The name of the variable
         * @return A copy of that variable
         * @throws GsccCodeException If that variable doesn't exist
         */
        public DataType getVariable(String name) throws GsccCodeException;
//returns result

        public DataType getAttrib(String name, String attrName) throws
GsccCodeException;

        /**
         * In the initial walk, registers a block with the interpreter to be
later matched against input.
         * @param regex regular expression to be checked on text input
         * @param type line or global
         * @param node node of the body of the regex block
         */
        public void registerRegexBlock(String regex, String type,
antlr.collections.AST node); //returns nothing

        /**
         * If the interpeter has been set up, runs it against input
         */
        public void runInput(java.io.BufferedReader in, AST program) throws
Exception;
```

```
        /**
         * Increases the symbol table by one level.  used for while and if
         *
         */
        public void startBlock();

        /**
         * Decreases the symbol table by one level.
         *
         */
        public void endBlock();

        /**
         * Sets up a while loop.
         * @param node The node of the body.  Presently unused
         * @return id of the loop
         */
        public int whileInit(AST node);

        /**
         * Tells the walker if execution can proceed.
         * @param id of the loop
         * @return Returns true unless break or return is set
         */
        public boolean whileCanProceed(int id);

        /**
         * Ends exectuion of a while loop.
         * @param id of the loop
         */
        public void whileEnd(int id);

        /**
         * Sets the break status.
         * @param id of the loop
         */
        public void breakWhile(int id);

        /**
         * Sets function return, which has the effect of halting execution
as needed
         * @param value to be returned
         */
        public void setReturn(DataType value);

}
```

8.7.7 LineReader.java

```
package interpreter;
import java.io.*;
/**
 *
 * @author ericgar
 *
```

```
 */


public class LineReader{
        private BufferedReader source;

        public LineReader(BufferedReader in) {
                source = in;
        }

        public String readLine() throws IOException{
                StringBuffer buffer = new StringBuffer();

                int endChar;

                //read until we get to the end of file or a line terminator
                while((endChar = source.read()) != -1 && endChar != 10 &&
endChar != 11 && endChar !=12){
                        buffer.append((char)endChar);
                }

                //if we found the end of file and we didn't read anything
                if(endChar == -1 && buffer.length() == 0){
                        return null;
                } else if (endChar == -1 ){ //otherwise, if we found the
end of file and *did* read something
                        return buffer.toString();
                }

                //if we have a \r we append, but keep going, because \r
could be followed by \n
                if(endChar == 13){
                        buffer.append((char) endChar);
                } else if(endChar == 10){ // if we have a \n, we add it and
stop
                        buffer.append((char) endChar);
                        return buffer.toString();
                }
                else { // if we have neither, then we stop here
                        return buffer.toString();
                }

                //we have to continue because \r could be followed by a
line feed

                //if we pick up a real character, we have to be able to
revert to this mark
                source.mark(0);
                int nextChar = source.read();
                if(nextChar == -1){ //if we've reached the end of file, we
return the line
                        return buffer.toString();
                }
                //if the next character is a \n we add it and return
                if(nextChar == 10){
                        buffer.append((char) endChar);
                        return buffer.toString();
```

```
                } else { //otherwise, we picked up a real character.
pretend we didn't and return the line.
                        source.reset();
                        return buffer.toString();
                }
        }
}
```

8.7.8 SymbolTable.java

```
package interpreter;
import java.util.Hashtable;
import types.DataType;
/**
 * SymbolTable represents the symbols internally, to ensure the scope is
correct
 * @author Casey Callendrello
 *
 */

public class SymbolTable extends Hashtable<String, DataType> {
        private SymbolTable parent;
        public SymbolTable previous;


        /**
         * The constructor for a root table
         *
         */
        public SymbolTable(){
                this(null, null);
        }

        /**
         * The constructor for a new table
         * @param parent The parent symbol table.
         * @param previous the previous symbol table
         */

        public SymbolTable(SymbolTable parent, SymbolTable previous){
                super();
                this.parent = parent;
                this.previous = previous;
        }

        /**
         * A method to determine if the symbol table has the variable
         * @param name The ID of the variable
         * @return Returns true if the variable exists in this table or any
of the parents
         */
        public boolean containsKey(String name){
                if(parent == null){
                        return super.containsKey(name);
                }
```

```
                else{
                        if(super.containsKey(name)){
                                return true;
                        }
                        return parent.containsKey(name);
                }
        }

        /** Returns the datatype from that name
         * @param name The name of the symbol
         * @return The symbol
         */
        public DataType get(String name){
                if(!containsKey(name)){
                        throw new IllegalArgumentException();
                }
                DataType ret = super.get(name);
                if(ret == null){
                        return parent.get(name);
                }
                return ret;

        }


}
```

8.8 Testing Files

-- please see attached sheets for complete listing; testing files not indexed. --

```
/*
Use input from file Mary.txt
ok so this test will be for scope
ie: things should only exist within the blocks they were declared in
and global works correctly as global
*/

set $globalstr, "I rule the world";
set #globalnum, #length($globalstr);

func $foo($one, #two)
{
        if ($one == $globalstr)
        {
                set $who, "globalstr rules";
                print $who + $substr($globalstr, 6, #globalnum - 1) + "\n";
        }
        else if ($who == "globalstr rules")
        {
                prerr "globalstr shouldn't be ruling the world\n";
        }
        else if (#two == #magicnum)
        {
                prerr "oof, the magicnum should have performed a disappearing act\n";
        }
        else
        {
                print $one + $substr($globalstr, 6, #globalnum - 1) + "\n";
                print "Today's lucky number is " + #two + "\n";
        }
}

func #foo($three, #four)
{
        if ($three == $one)
        {
                print "Oddly enough, I'm expecting this to print.\n";
        }
        else
        {
                prerr $three + "\n";
        }
        if (#four == #two)
        {
                print "And again, I'm thinking this will print.\n";
        }
        else
        {
                prerr #four + "\n";
        }
}

[Mary] line {
        set $ruler, @match + " rules";
        set #magicnum, 7;
        $foo($ruler, #magicnum);
}

[snow] line {
        if ($ruler == @match + " rules")
        {
                prerr "Mary has stepped outside of her bounds\n";
        }
        else if (@line.length == #magicnum)
        {
                prerr "This is some crazy magic.\n";
        }
        else if (@match == $who)
        {
                prerr "We don't even know who is who yet\n";
```

```
        }
        else
        {
                set $ruler, $globalstr;
                set #itsmagical, #globalnum;
                $foo($ruler, #itsmagical);
                #foo($one, #two);
        }
}

//I'm predicting this output
/*

*/
```

```
/*
Use input from file Mary.txt
ok so this test will be for scope
ie: things should only exist within the blocks they were declared in
and global works correctly as global
*/

set $globalstr, "I rule the world";
set #globalnum, #length($globalstr);

func $foo($one, #two)
{
        if ($one == $globalstr)
        {
                set $who, "globalstr rules";
                print $who + $substr($globalstr, 6, #globalnum - 1) + "\n";
        }
        else
        {
                print $one + $substr($globalstr, 6, #globalnum - 1) + "\n";
                print "Today's lucky number is " + #two + "\n";
        }
/*
        if ($who == "globalstr rules")
        {
                prerr "globalstr shouldn't be ruling the world\n";
        }*/
        if (#two == #magicnum)
        {
                prerr "oof, the magicnum should have performed a disappearing act\n";
        }

}

func #foo($three, #four)
{

        if ($three == $one)
        {
                print "Oddly enough, I'm expecting this to print.\n";
        }
        else
        {
                prerr $three + "\n";
        }
        if (#four == #two)
        {
                print "And again, I'm thinking this will print.\n";
        }
        else
        {
                prerr #four + "\n";
        }
}

[Mary] line {
        set $ruler, @match + " rules";
        set #magicnum, 7;
        $foo($ruler, #magicnum);
}

[snow] line {
        if ($ruler == @match + " rules")
        {
                prerr "Mary has stepped outside of her bounds\n";
        }
//      if (@line.length == #magicnum)
//      {
//              prerr "This is some crazy magic.\n";
//      }
```

```
/* this correctly generates an error
        if (@match == $who)
        {
                prerr "We don't even know who is who yet\n";
        }
*/
        set $ruler, $globalstr;
        set #itsmagical, #globalnum;
        $foo($ruler, #itsmagical);
/*      #foo($one, #two);*/
}

//I'm predicting this output
/*

*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        print $substr("test string", 0, 6) + "\n";
}

[snow] line {
        print #length("test string") + "\n";
}

//I'm predicting this output
/*
test s
11
*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        set $var, "test string";
        print $substr($var, 0, 6) + "\n";
}

[snow] line {
        set $variable, "test string";
        print #length($variable) + "\n";
}

//I'm predicting this output
/*
test s
11
*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        set $var, "test string";
        print $substr(1234567, 0, 6) + "\n";
}

[snow] line {
        set $variable, "test string";
        print #length(1234567) + "\n";
}

//I'm predicting this output
/*
123456
7
*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        set #var, 1234567;
        print $substr(#var, 0, 6) + "\n";
}

[snow] line {
        set #variable, 1234567;
        print #length(#variable) + "\n";
}

//I'm predicting this output
/*
123456
7
*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        set #var, 1234567;
        print $substr("hi" + #var, 0, 6) + "\n";
}

[snow] line {
        set #variable, 1234567;
        print #length("hi" + #variable) + "\n";
}

//I'm predicting this output
/*
hi1234
9
*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        set #var, 1234567;
        print $substr("hi" + 123, 3, 7) + "\n";
}

[snow] line {
        set #variable, 1234567;
        print #length("hi" + 123) + "\n";
}

//I'm predicting this output
/*
23
5
*/
```

```
/*
Use input from file Mary.txt
these tests are for the 2 built in functions $substr and #length
*/

[Mary] line {
        set #var, 1234567;
        print $substr(27 + 123, 1, 2) + "\n";
}

[snow] line {
        set #variable, 1234567;
        print #length(27 + 123) + "\n";
}

//I'm predicting this output
/*
50
3
*/
```

```
[Mary] line {
        print 0 + "1" + "\n";
        print 0 + #parse("1") + "\n";
        print 0 + #parse("a") + "\n";
}
```

```
/*
delete tests, all will be done on the input from file Mary.txt
*/

[i] line {
        set $test, "this is a sentance\n";
        delete $test, 1, 1;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        delete $testing, 1+2, 3;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
test differnt combinations of integers and variables
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 1;
        delete $test, #int1, 1;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        delete $testing, 3, 1+2;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
test differnt combinations of integers and variables
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 1;
        set #int2, 1;
        delete $test, #int1, #int2;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        delete $testing, 1+2, 2+1;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
test differnt combinations of integers and variables
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 1;
        delete $test, 1, #int1;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        delete $testing, 2+1, 1+2;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
last thing to test would be things like 1+#int
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 1;
        set #int2, 1;
        delete $test, 0+#int1, #int2-0;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        set #integer, 1;
        delete $testing, #integer*3, 2+1;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
last thing to test would be things like 1+#int
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 4;
        set #int2, 2;
        delete $test, #int1/4, #int2-1;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        set #integer, 1;
        delete $testing, 3, #integer*3;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
last thing to test would be things like 1+#int
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 4;
        set #int2, 1;
        delete $test, 4/#int1, 2-#int2;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        set #integer, 1;
        delete $testing, #integer*3, #integer*3;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
and now assuming all those evaluations worked, expand them to longer ones
not gonna write a lot of tests for this since if it works, it was already
tested by the ones in the SET section
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int1, 4;
        set #int2, 5;
        delete $test, (#int1/2)-1, (2-2)+(#int2-3)/2;
        print $test;
}

[snow] line {
        /* so heres a thought, i looked at the lrm and the format is
        string, int, int but do we evaluate expressions for it before
        checking? ie: this example below */
        set $testing, "testing for operators\n";
        set #integer, 1;
        delete $testing, #integer*(1+1+1), ((#integer-1)+1)*3;
        print $testing;
}

//I'm predicting this output
/*
tis is a sentance
tis is a sentance
tesg for operators
*/
```

```
/*
Use input from file Mary.txt
assuming you've tested the if and elseif files,
all we have to do is make sure else is entered when it should be
and not entered when it shouldn't be
*/

[Mary] line {
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else
        {
                print "else\n";
        }
}

[snow] line {
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else
        {
                print "else again\n";
        }
}

//I'm predicting this output
/*
else
else again
*/
```

```
/*
Use input from file Mary.txt
assuming you've tested the if and elseif files,
all we have to do is make sure else is entered when it should be
and not entered when it shouldn't be
*/

[Mary] line {
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else
        {
                print "else\n";
        }
}

[snow] line {
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else
        {
                print "else again\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
}//*
}//*
*//*
```

```
/*
Use input from file Mary.txt
enter if, skip else if
*/

[Mary] line {
        set #num, 1;
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num < 6 && 5 > #num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 1;
        if (1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number < 1 || (#num < 6 && 5 > #num))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
1-5, skip if, enter else if
6-10, enter if, skip else if
11-15, skip both
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (!#num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        print "outside\n";
}

[snow] line {
        set #number, 0;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number)  /* there was a typo, there should have not been a ! */
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        print "outside again\n";
}

//I'm predicting this output
/*
outside
outside again
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (!(#num < 5))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        print "outside\n";
}

[snow] line {
        set #number, 6;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (!(#number > 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        print "outside again\n";
}

//I'm predicting this output
/*
outside
outside again
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (!(#num <= 5))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        print "outside\n";
}

[snow] line {
        set #number, 5;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (!(#number >= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        print "outside again\n";
}

//I'm predicting this output
/*
outside
outside again
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (!(#num != 5))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        print "outside\n";
}

[snow] line {
        set #number, 5;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (!(#number == 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        print "outside again\n";
}

//I'm predicting this output
/*
outside
outside again
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (!(#num < 6 && 5 > #num))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        print "outside\n";
}

[snow] line {
        set #number, 1;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (!(#number < 1 || (#num < 6 && 5 > #num)))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        print "outside again\n";
}

//I'm predicting this output
/*
outside
outside again
*/
```

```
/*
Use input from file Mary.txt
all we have to do is test that else if works will
all the different operators since if all if tests worked,
all the operators should work correctly, so no need for multiple tests
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num < 5)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 6;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number > 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hello
*/
```

```
/*
Use input from file Mary.txt
all we have to do is test that else if works will
all the different operators since if all if tests worked,
all the operators should work correctly, so no need for multiple tests
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num <= 5)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 5;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number >= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hello
*/
```

```
/*
Use input from file Mary.txt
all we have to do is test that else if works will
all the different operators since if all if tests worked,
all the operators should work correctly, so no need for multiple tests
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num != 5)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 5;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number == 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hello
*/
```

```
/*
Use input from file Mary.txt
all we have to do is test that else if works will
all the different operators since if all if tests worked,
all the operators should work correctly, so no need for multiple tests
*/

[Mary] line {
        set #num, 1;
        if (0)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num < 6 && 5 > #num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 1;
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number < 1 || (#num < 6 && 5 > #num))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hello
*/
```

```
/*
Use input from file Mary.txt
1-5 the else if should always be entered
6-10 the if should be entered, else if skipped
*/

[Mary] line {
        set #num, 1;
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 0;
        if (1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
entering the if, skipping the else if
*/

[Mary] line {
        set #num, 1;
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num < 5)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 6;
        if (1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number > 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
enter if, skip else if
*/

[Mary] line {
        set #num, 1;
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num <= 5)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 5;
        if (1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number >= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
enter if, skip else if
*/

[Mary] line {
        set #num, 1;
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        else if (#num != 5)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #number, 5;
        if (1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        else if (#number == 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
first set of tests will be on functions that don't return anything
these will test the passing of args, and that you actually enter them
*/

func $foo()
{
        print "$enter\n";
}

func #foo()
{
        print "#enter\n";
}

[Mary] line {
        $foo();
        print "exit\n";
}

[snow] line {
        #foo();
        print "exit\n";
}

//I'm predicting this output
/*
$enter
exit
#enter
exit
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        set $bleh, #two;
        return $one + $bleh;
}

func #foo($one, $three, #four, #two)
{
        return #four - #two;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
te36
-22
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        set $bleh, #two;
        return #two;
}

func #foo($one, $three, #four, #two)
{
        return #two/#four;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
6
12
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        set $bleh, #two;
        return 1337;
}

func #foo($one, $three, #four, #two)
{
        return (#two/#four) + 100;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
1337
112
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        set $bleh, #two;
        return 1337 + #two;
}

func #foo($one, $three, #four, #two)
{
        return (#two/#four) + 100*2;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
1343
212
*/
```

```
/*
Use input from file Mary.txt
first set of tests will be on functions that don't return anything
these will test the passing of args, and that you actually enter them
*/

func $foo($enter)
{
        print $enter + "\n";
}

func #foo(#enter)
{
        print #enter + "\n";
}

[Mary] line {
        $foo("test");
        print "exit\n";
}

[snow] line {
        #foo(5);
        print "exit\n";
}

//I'm predicting this output
/*
test
exit
5
exit
*/
```

```
/*
Use input from file Mary.txt
first set of tests will be on functions that don't return anything
these will test the passing of args, and that you actually enter them
*/

func $foo(#enter)
{
        print #enter + "\n";
}

func #foo($enter)
{
        print $enter + "\n";
}

[Mary] line {
        $foo(5);
        print "exit\n";
}

[snow] line {
        #foo("test");
        print "exit\n";
}

//I'm predicting this output
/*
5
exit
test
exit
*/
```

```
/*
Use input from file Mary.txt
first set of tests will be on functions that don't return anything
these will test the passing of args, and that you actually enter them
*/

func $foo($one, #two, $three, #four)
{
        print $one + " " + #two + " " + $three + " " + #four + "\n";
}

func #foo($one, #two, $three, #four)
{
        print $one + " " + #two + " " + $three + " " + #four + "\n";
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        $foo("te" + "st", 1*5, $arg, #arg);
        print "exit\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        #foo("te" + "sts", 2-1, $args, #args);
        print "exit\n";
}

//I'm predicting this output
/*
test 5 hi 2
exit
tests 1 hello 4
exit
*/
```

```
/*
Use input from file Mary.txt
first set of tests will be on functions that don't return anything
these will test the passing of args, and that you actually enter them
*/

func $foo($one, #two, #four, $three)
{
        print $one + " " + #two + " " + $three + " " + #four + "\n";
}

func #foo($one, $three, #four, #two)
{
        print $one + " " + #two + " " + $three + " " + #four + "\n";
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        $foo("te" + 3, 1*2*3, #arg-6, $arg + 5);
        print "exit\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        #foo("te" + 4, $args + 3, #args/2, 2*3*4);
        print "exit\n";
}

//I'm predicting this output
/*
te3 6 hi5 -4
exit
te4 24 hello3 2
exit
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        return "testing";
}

func #foo($one, $three, #four, #two)
{
        return 123;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
testing
123
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        return $one;
}

func #foo($one, $three, #four, #two)
{
        return #two;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
te3
24
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        return "test" + "ing";
}

func #foo($one, $three, #four, #two)
{
        return 100 + 23;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
testing
123
*/
```

```
/*
Use input from file Mary.txt
tests 1-5 were for non-returns
tests 6+ are for those that return
*/

func $foo($one, #two, #four, $three)
{
        return $one + "test";
}

func #foo($one, $three, #four, #two)
{
        return 100 + #two;
}

[Mary] line {
        set $arg, "hi";
        set #arg, 2;
        print $foo("te" + 3, 1*2*3, #arg-6, $arg + 5) + "\n";
}

[snow] line {
        set $args, "hello";
        set #args, 4;
        print #foo("te" + 4, $args + 3, #args/2, 2*3*4) + "\n";
}

//I'm predicting this output
/*
te3test
124
*/
```

```
/*
Use input from file Mary.txt
all of the if tests will also be testing for the functionality
of the different operators <, >, <=, >=, ==, !=, &&, and ||
*/

[Mary] line {
        if (1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 1;
        if (#num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 0;
        if (#number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num-5+2 < 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb*5+4 < 2+3+3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 < #number+#numbe/3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3*2*4 < #number*#numbe*10))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
testing if and operators at same time
5-10 was with <
now 11-16 will be the same but switch to >
*/

[Mary] line {
        if (1 > 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num > 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (5 > 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (#number > 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb > #num)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (5 > #numb)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (#numbe > #number)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (1 > #numbe)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >
*/

[Mary] line {
        if (!(1 > 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!(#num > 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(5 > 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (!(#number > 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb*#num > 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (#numb > 2+4)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (5 > #number+#numbe)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (5-3 > #number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num > 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb > 2+4))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 > #number+#numbe))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3 > #number))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
last one for >
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num-5+2 > 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb*5+4 > 2+3+3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 > #number+#numbe/3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3*2*4 > #number*#numbe*10))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
testing if and operators at same time
5-10 was with <
11-16 was with >
17-22 will be with <=
*/

[Mary] line {
        if (1 <= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num <= 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (5 <= 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (#number <= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb <= #num)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (5 <= #numb)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (#numbe <= #number)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (1 <= #numbe)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        if (!(1 <= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!(#num <= 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(5 <= 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (!(#number <= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        if (10)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (6-6)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        // so what exactly do strings get converted to?
        // are all of them equal to 1, or will the 0's get converted to 0
        // the output should clarify this if an extra line of "hello" is printed
        // strings are converted if they are forced.   This is a subtle detail
        set $number, 0;
        if ($number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb*#num <= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (#numb <= 2+4)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (5 <= #number+#numbe)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (5-3 <= #number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num <= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb <= 2+4))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 <= #number+#numbe))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3 <= #number))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num-5+2 <= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb*5+4 <= 2+3+3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 <= #number+#numbe/3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3*2*4 <= #number*#numbe*10))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
testing if and operators at same time
5-10 was with <
11-16 was with >
17-22 was with <=
23-28 will be with >=
*/

[Mary] line {
        if (1 >= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num >= 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (5 >= 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (#number >= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >=
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb >= #num)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (5 >= #numb)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (#numbe >= #number)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (1 >= #numbe)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >=
*/

[Mary] line {
        if (!(1 >= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!(#num >= 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(5 >= 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (!(#number >= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >=
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb*#num >= 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (#numb >= 2+4)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (5 >= #number+#numbe)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (5-3 >= #number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
still on >=
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num >= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb >= 2+4))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 >= #number+#numbe))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3 >= #number))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
last one for >=
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num-5+2 >= 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb*5+4 >= 2+3+3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 >= #number+#numbe/3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3*2*4 >= #number*#numbe*10))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
testing if and operators at same time
5-10 was with <
11-16 was with >
17-22 was with <=
23-28 was with >=
29-34 will be with ==
*/

[Mary] line {
        if (5 == 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num == 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (5 == 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (#number == 1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        if (!(1))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 1;
        if (!#num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(0))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 0;
        if (!#number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb == #num)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (5 == #numb)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (#numbe == #number)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (1 == #numbe)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        if (!(1 == 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!(#num == 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(5 == 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (!(#number == 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hi
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb*#num == 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (#numb == 2+4)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (5 == #number+#numbe)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (5-3 == #number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num == 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb == 2+4))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 == #number+#numbe))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3 == #number))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num-5+2 == 5)) /* this is true ! ( 2 == 5) */
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb*5+4 == 2+3+4)) /* this is false: ! (9 == 9) */
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 == #number+#numbe/3))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3*2*4 == #number*#numbe*10))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
*/

//Casey is predicting this output:
/*
hello
hi
hello
*/
```

```
/*
Use input from file Mary.txt
testing if and operators at same time
5-10 was with <
11-16 was with >
17-22 was with <=
23-28 was with >=
29-34 was with ==
35-40 will be with !=
*/

[Mary] line {
        if (5 != 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num != 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (5 != 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (#number != 1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb != #num)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (5 != #numb)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (#numbe != #number)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (1 != #numbe)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hi
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        if (!(1 != 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!(#num != 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(5 != 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (!(#number != 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb*#num != 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (#numb != 2+4)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (5 != #number+#numbe)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (5-3 != #number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num != 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb != 2+4))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 != #number+#numbe))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3 != #number))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
*/
```

```
/*
Use input from file Mary.txt
*/

[Mary] line {
        if (!(10))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!#num)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(6-6))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        // this should give the opposite result of the question
        // posed in the second test case
        /* since there is nothin to cast this as a number, it returns false.  */
        set $number, 0;
        if (!$number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num-5+2 != 5)) /* this is false... 2 == 5 */
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb*5+4 != 2+3+4)) /* this is true... */
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 != #number+#numbe/3)) /* this is false */
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3*2*4 != #number*#numbe*10)) /* this is false */
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hello
*/

//this is wrong, the output should just be
/*
hi
*/
```

```
/*
Use input from file Mary.txt
all of the if tests will also be testing for the functionality
both && and || will be tested at the same time in tests 41+
*/

[Mary] line {
        if (1 && 1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 1;
        if (#num && 0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (0 || 0)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 0;
        if (#number || 1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
all of the if tests will also be testing for the functionality
both && and || will be tested at the same time in tests 41+
*/

[Mary] line {
        if (1<5 && 5>1)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 1;
        if (#num>=6 && 0<=11)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (1<5 || 5>1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 0;
        if (#number>=6 || 0<=11)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hello
*/
```

```
/*
Use input from file Mary.txt
all of the if tests will also be testing for the functionality
both && and || will be tested at the same time in tests 41+
*/

[Mary] line {
        if (2==2 && 3!=4)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 1;
        if (!(1) && !(0<=11))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (1==5 || 5!=1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 0;
        if (!(#number>=6) || !(0==11))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hi
hello
*/
```

```
/*
if tests, all will be done on the input from file Mary.txt
*/

[Mary] line {
        set $test, "this is a sentance\n";
        set $n, 5;

        if($n){
                print "YES single number variable in if works\n";
        } else {
                print "NO single number variable in if does not work\n";
        }


        if(1){
                print "YES single number in if works\n";
        } else {
                        print "NO single number in if\n";
        }

        if(0){
                print "NO else doens't work\n";

        } else {
                print "YES else works\n";
        }

        if(0){
                        print "NO elseif doesn't work\n";
        } else if (1) {
                print "YES elseif works\n";
        }

        if($test){
                print "YES single string in if works \n";
        } else{
                print "NO single string in if doesn't work \n";
        }

        set $test2, "";

        if($test2){
                print "NO single empty string in if doesn't work\n";
        } else {
                print "YES single empty string in if does work\n";
        }

        if(-1){
                print "YES negative in if does work\n";
        } else {
                print "NO negative in if doesn't work\n";
        }

        if(1+1){
                print "YES expression in if works\n";
        } else {
                print "NO expression in if doesn't work\n";
        }

        if(3 > 2){
                print "YES gt works\n";
        } else {
                print "NO gt doesn't work\n";
        }

        if(2 < 3){
                print "YES lt than works\n";
        } else {
                print "NO lt doesn't work\n";
```

```
        }

        if(2 <= 3){
                print "YES lteq works\n";
        } else {
                print "NO lteq doesn't work\n";
        }

        if(2 <= 2){
                print "YES lteq works\n";
        } else {
                print "NO lteq doesn't work\n";
        }

        if(3 >= 2){
                print "YES gteq works\n";
        } else {
                print "NO gteq doesn't work\n";
        }

        if(3 >= 2){
                print "YES gteq works\n";
        } else {
                print "NO gteq doesn't work\n";
        }

        if(2 == 2){
                print "YES eq works\n";
        } else {
                print "NO eq doesn't work\n";
        }

        if(2 != 2){
                print "NO neq doesn't works\n";
        } else {
                print "YES neq does work\n";
        }

        if(1 != 2){
                print "YES neq\n";
        } else {
                print "NO neq \n";
        }

        if((1==1) && (2==2)){
                print "YES && \n";
        } else {
                print "NO && \n";
        }

        if((0==1) && (2==2)){
                print "NO && \n";
        } else {
                print "YES && \n";
        }

        if((0==1) && (1==2)){
                print "NO && \n";
        } else {
                print "YES && \n";
        }

        if((0==1) || (2==2)){
                print "YES || \n";
        } else {
                print "NO || \n";
        }

        if((1==1) || (1==2)){
                print "YES || \n";
```

```
	} else {
		print "NO || \n";
	}

	if((1==2) || (1==2)){
		print "NO || \n";
	} else {
		print "YES || \n";
	}

	set #h, 1 > 2;
	if(#h == 1){
		print "NO value returned from gt \n";
	} else  if(#h == 0){
		print "YES value returned from gt \n";
	} else{
		print "NO value returned from gt \n";
	}

	set #h, 3 > 2;
	if(#h == 1){
		print "YES value returned from gt \n";
	} else {
		print "NO value returned from gt \n";
	}

	set #h, 1 < 2;
	if(#h == 1){
		print "YES value returned from lt \n";
	} else  if(#h == 0){
		print "NO value returned from lt \n";
	} else{
		print "NO value returned from lt \n";
	}

	set #h, 3 < 2;
	if(#h == 1){
		print "NO value returned from lt \n";
	} else if(#h == 0) {
		print "YES value returned from lt \n";
	} else{
		print "NO value returned from gt \n";
	}

	set #h, 1;

	if(! #h) {
		print "NO value returned from ! \n";
	} else {
		print "YES value returned from ! \n";
	}

	if( (0 || 0) && 1){
		print "NO (0 || 0) && 1 \n";
	} else {
		print "YES (0 || 0) && 1 \n";
	}

	if( (0 || 1) && 1){
		print "YES (0 || 1) && 1 \n";
	} else {
		print "NO (0 || 1) && 1 \n";
	}

	if( 0 || 0 && 1){
		print "NO 0 || 0 && 1 \n";
	} else {
		print "YES 0 || 0 && 1 \n";
	}
```

```
        if( 0 && 0 && 1){
                print "NO  0 && 0 && 1 \n";
        } else {
                print "YES  0 && 0 && 1 \n";
        }

        if( 1 && 1 && 1){
                print "YES  1 && 1 && 1 \n";
        } else {
                print "NO  1 && 1 && 1 \n";
        }

        if( 1 || 0 || 0){
                print "YES   1 || 0 || 0 \n";
        } else {
                print "NO   1 || 0 || 0 \n";
        }

        if( 0 || 0 || 0){
                print "NO 0 || 0 || 0 \n";
        } else {
                print "YES 0 || 0 || 0 \n";
        }




}
```

```
/*
if tests, all will be done on the input from file Mary.txt
*/

[Mary] line {

        set $test2, "";

        /*if($test2){
                print "NO single empty string in if doesn't work\n";
        } else {
                print "YES single empty string in if does work\n";
        }*/

        if(-1){
                print "YES negative in if does work\n";
        } else {
                print "NO negative in if doesn't work\n";
        }

}
```

```
/*
Use input from file Mary.txt
testing if and operators at same time
*/

[Mary] line {
        if (1 < 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (#num < 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (5 < 1)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (#number < 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb < #num)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (5 < #numb)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (#numbe < #number)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (1 < #numbe)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        if (!(1 < 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        set #num, 5;
        if (!(#num < 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        if (!(5 < 1))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        set #number, 1;
        if (!(#number < 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (#numb*#num < 5)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (#numb < 2+4)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (5 < #number+#numbe)
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (5-3 < #number)
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hi
hi
*/
```

```
/*
Use input from file Mary.txt
still testing if and operators at same time
*/

[Mary] line {
        set #numb, 1;
        set #num, 5;
        if (!(#numb*#num < 5))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
        if (!(#numb < 2+4))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
}

[snow] line {
        set #numbe, 5;
        set #number, 1;
        if (!(5 < #number+#numbe))
        {
                set $hello, "hi";
                print $hello + "\n";
        }
        if (!(5-3 < #number))
        {
                set $hello, "hello";
                print $hello + "\n";
        }
}

//I'm predicting this output
/*
hello
hello
*/
```

```
/*
ok testing insert now, same question about the expressions for int - is it allowed?
this'll follow the same format as the others, i'll try not to change the output
*/

[i] line {
        set $test, "this is a sentance\n";
        insert $test, 5, "ladida";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        insert $testing, 1/1, "bleh";
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
ok testing insert now, same question about the expressions for int - is it allowed?
this'll follow the same format as the others, i'll try not to change the output
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        insert $test, #int, "ladida";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        insert $testing, 0+1-0, "bleh";
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
ok testing insert now, same question about the expressions for int - is it allowed?
this'll follow the same format as the others, i'll try not to change the output
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        insert $test, #int, "lad" + "ida";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        insert $testing, 1*1-0, "bleh";
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
ok testing insert now, same question about the expressions for int - is it allowed?
this'll follow the same format as the others, i'll try not to change the output
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        set $str, "lad";
        insert $test, #int, $str + "ida";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        insert $testing, (#integer-3)/2, "bleh";
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
ok testing insert now, same question about the expressions for int - is it allowed?
this'll follow the same format as the others, i'll try not to change the output
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        set $str, "ida";
        insert $test, #int, "lad" + $str;
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        set $bleh, "bleh";
        insert $testing, (#integer-3)/2, $bleh;
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
ok testing insert now, same question about the expressions for int - is it allowed?
this'll follow the same format as the others, i'll try not to change the output
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        set $str, "ida";
        set $str2, "lad";
        insert $test, #int, $str2 + $str;
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        set $bleh, "bleh";
        insert $testing, (#integer*2)/10, $bleh;
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
from this test on, i'm testing longer expression evaluations
everything else should have been tested for
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        set $str, "di";
        set $str2, "la";
        insert $test, #int, $str2 + $str + "da";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        set $bleh, "bleh";
        insert $testing, (#integer*2)/10, $bleh;
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
```

```
/*
from this test on, i'm testing longer expression evaluations
everything else should have been tested for
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        set $str, "di";
        set $str2, "da";
        insert $test, #int, "la" + $str + $str2;
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        set $bleh, "b";
        insert $testing, #integer, $bleh + "l" + "e" + "h";
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/
//this is wrong, bleh is inserted to index 5
/*
this ladidais a sentance
this ladidais a sentance
testiblehng for operators
*/
```

```
/*
from this test on, i'm testing longer expression evaluations
everything else should have been tested for
*/

[i] line {
        set $test, "this is a sentance\n";
        set #int, 5;
        set $str, "di";
        set $str2, "da";
        set $str3, "la";
        insert $test, #int, $str3 + $str + $str2;
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        set $bleh, "b";
        insert $testing, #integer, $bleh + "l" + "e" + "h";
        print $testing;
}

//I'm predicting this output
/*
this ladidais a sentance
this ladidais a sentance
tblehesting for operators
*/

//The above is wrong, this is the correct output/
/*
this ladidais a sentance
this ladidais a sentance
testiblehng for operators
*/
```

```
[Mary] line {
        print #length("1234") + "\n";
        print (#length("5-2") - 1) + "\n";

}
```

```
[a] line {
        print "print line: ";
        print @line;
        print "Line: " + @line.line;
        print "\nStart: " + @line.start;
        print "\nEnd: " + @line.end;
        print "\nLength: " + @line.length;
        print "\nprint match: ";
        print @match + "\n";
        print "Line: " + @match.line;
        print "\nStart: " + @match.start;
        print "\nEnd: " + @match.end;
        print "\nLength: " + @match.length + "\n";

}
```

```
/*

Exception in thread "main" java.lang.NullPointerException
        at gscc.main(gscc.java:89)

*/


[l.] line {
        print "print line: ";
        print @line;
        print "Line: " + @line.line + "\n";

}
```

```
[1.] line {
        print @match + "\n";
        set @match, "ABC";
        print @line;
        set @line, "i am setting the line\n";
        print @line;
}
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        set @line, @line;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        set @match, @match;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        replace @line, 0, @line;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        replace @match, 0, @match;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        replace @line, 7, "replacement";
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        replace @match, 2, "replacement";
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary hareplacementlamb

1
0
22
23
Mareplacement
1
0
12
13
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        replace @line, 7, 1234;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        replace @match, 2, 1234;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary ha1234little lamb

1
0
22
23
Ma1234
1
0
5
6
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        replace @line, 7, @match;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        replace @match, 2, @line;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary haMarylittle lamb

1
0
22
23
MaMary haMarylittle lamb

1
0
24
25
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        print $substr(@line, 3, 4) + "\n";
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        print $substr(@match, 2, 3) + "\n";
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
y ha
Mary had a little lamb

1
0
22
23
ry
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        print #length(@line) + "\n";
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        print #length(@match) + "\n";
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
23
Mary had a little lamb

1
0
22
23
4
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

// for funcs, just need to test that they get converted correct and are passed
// after that, they are no longer locations
func $foo($one, $two, $three, $four, $five)
{
        print $one + "\n";
        print $two + "\n";
        print $three + "\n";
        print $four + "\n";
        print $five + "\n";
}

func #foo($one, $two, $three, $four, $five)
{
        print $one + "\n";
        print $two + "\n";
        print $three + "\n";
        print $four + "\n";
        print $five + "\n";
}

[Mary] line {
        $foo(@line, @line.line, @line.start, @line.end, @line.length);
        #foo(@line, @line.line, @line.start, @line.end, @line.length);
}

[Mary] line {
        $foo(@match, @match.line, @match.start, @match.end, @match.length);
        #foo(@match, @match.line, @match.start, @match.end, @match.length);
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

// for funcs, just need to test that they get converted correct and are passed
// after that, they are no longer locations
func $foo($one, #two, #three, #four, #five)
{
        print $one + "\n";
        print #two + "\n";
        print #three + "\n";
        print #four + "\n";
        print #five + "\n";
}

func #foo($one, #two, #three, #four, #five)
{
        print $one + "\n";
        print #two + "\n";
        print #three + "\n";
        print #four + "\n";
        print #five + "\n";
}

[Mary] line {
        $foo(@line, @line.line, @line.start, @line.end, @line.length);
        #foo(@line, @line.line, @line.start, @line.end, @line.length);
}

[Mary] line {
        $foo(@match, @match.line, @match.start, @match.end, @match.length);
        #foo(@match, @match.line, @match.start, @match.end, @match.length);
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        if (@line)
        {
                print @line + "\n";
                print @line.line + "\n";
                print @line.start + "\n";
                print @line.end + "\n";
                print @line.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

[Mary] line {
        if (@match)
        {
                print @match + "\n";
                print @match.line + "\n";
                print @match.start + "\n";
                print @match.end + "\n";
                print @match.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        set @line, "change the line\n";
        print @line;
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[line] line {
        set @match, "change the match";
        print @line;
        //print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
change the line
1
0
14
15
change the match
1
0
15
16
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        if (!@line)
        {
                print @line + "\n";
                print @line.line + "\n";
                print @line.start + "\n";
                print @line.end + "\n";
                print @line.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

[Mary] line {
        if (!@match)
        {
                print @match + "\n";
                print @match.line + "\n";
                print @match.start + "\n";
                print @match.end + "\n";
                print @match.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

//I'm predicting this output
/*
error
error
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        if (!@line)
        {
                print "shouldn't go here\n";
        }
        else if (@line)
        {
                print @line + "\n";
                print @line.line + "\n";
                print @line.start + "\n";
                print @line.end + "\n";
                print @line.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

[Mary] line {
        if (!@match)
        {
                print "shouldn't go here\n";
        }
        else if (@match)
        {
                print @match + "\n";
                print @match.line + "\n";
                print @match.start + "\n";
                print @match.end + "\n";
                print @match.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        if (@line == @match)
        {
                print "shouldn't go here\n";
        }
        else if (@line == @line)
        {
                print @line + "\n";
                print @line.line + "\n";
                print @line.start + "\n";
                print @line.end + "\n";
                print @line.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

[Mary] line {
        if (@match == @line)
        {
                print "shouldn't go here\n";
        }
        else if (@match == @match)
        {
                print @match + "\n";
                print @match.line + "\n";
                print @match.start + "\n";
                print @match.end + "\n";
                print @match.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

//I'm predicting this output
/*
Mary had a little lamb

1
0
22
23
Mary
1
0
3
4
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        if (!(@line == @match))
        {
                print "shouldn't go here\n";
        }
        else if (@line == @line)
        {
                print @line + "\n";
                print @line.line + "\n";
                print @line.start + "\n";
                print @line.end + "\n";
                print @line.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

[Mary] line {
        if (!(@match == @line))
        {
                print "shouldn't go here\n";
        }
        else if (@match == @match)
        {
                print @match + "\n";
                print @match.line + "\n";
                print @match.start + "\n";
                print @match.end + "\n";
                print @match.length + "\n";
        }
        else
        {
                print "error\n";
        }
}

//I'm predicting this output
/*
shouldn't go here
shouldn't go here
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        while (@line.length > 20)
        {
                set @line, $substr(@line, 0, @line.length - 1);
                print @line + "\n";
                print @line.line + "\n";
                print @line.start + "\n";
                print @line.end + "\n";
                print @line.length + "\n";
        }
}

[Mary] line {
        while (@match.length > 2)
        {
                set @match, $substr(@match, 0, @match.length - 1);
                print @match + "\n";
                print @match.line + "\n";
                print @match.start + "\n";
                print @match.end + "\n" ;
                print @match.length + "\n";
        }
}

//I'm predicting this output
/*
Mary had a little lamb
1
0
21
22
Mary had a little lam
1
0
20
21
Mary had a little la
1
0
19
20
Mar
1
0
2
3
Ma
1
0
1
2
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        set @line, 12345;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[123] line {
        set @match, 12345;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
12345
1
0
4
5
12345
1
0
4
5
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        print @line;
        delete @line, 5, 7;
        print @line;
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        delete @match, 0, 2;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary  a little lamb

1
0
19
20
y
1
0
0
1
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        set @match, @line;
        print @match;
        print @line;
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

[Mary] line {
        set @line, @match;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}



//I'm predicting this output
/*
Mary
1
0
3
4
Mary had a little lamb

1
0
22
23
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        insert @line, 4, "insert this";
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        insert @match, 1, "insert this";
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Maryinsert this had a little lamb

1
0
33
34
Minsert thisary
1
0
14
15
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        insert @line, 4, 1234;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        insert @match, 1, 1234;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
Mary1234 had a little lamb

1
0
26
27
M1234ary
1
0
7
8
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        insert @line, 4, @match;
        print @line + "\n";
        print @line.line + "\n";
        print @line.start + "\n";
        print @line.end + "\n";
        print @line.length + "\n";
}

[Mary] line {
        insert @match, 1, @line;
        print @match + "\n";
        print @match.line + "\n";
        print @match.start + "\n";
        print @match.end + "\n";
        print @match.length + "\n";
}

//I'm predicting this output
/*
MaryMary had a little lamb

1
0
26
27
MMaryMary had a little lamb
ary
1
0
26
27
*/
```

```
/*
Use input from file Mary.txt
testing both location variables at the same time
will use them wherever they should be able to be used
*/

[Mary] line {
        insert @line, 4, @line;
        prins + "\n" @line;
        prins.rins + "\n" @line;
```

```
/*
if tests, all will be done on the input from file Mary.txt
*/

[Mary] line {
        print "hello.  Hello." + -2 + "\n";
}
```

```
/*
if tests, all will be done on the input from file Mary.txt
*/

[Mary] line {
        prerr "Print to std. out\n";
}
```

```
[\n] line {
        print "YES \\n  \n";
}
```

```
/*
        why does it print twice?
*/


[\n] line {
        print "YES /\n\n";
}
```

```
/*
        why does it print twice?
*/


[\n] line {
        print @match == "\n";
        print "YES \n";
}
```

```
/*
        good.
*/


[$] line {
        print "YES \n";
}
```

```
/*

        first of global tests...


*/


[l.] global {
        print "print line: ";
        print @line;
}
```

```
/*

        first of global tests...


*/


[/*/] global {
        print "print line: ";
        print @line;
}
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

[Mary] line {
        print "line - all lines are once\n";
}

[Mary] global {
        print "global - this time just once\n";
}

[a] line {
        print "line - all lines are once\n";
}

[a] global {
        print "global - I should pop up 4x then 2x\n";
}

//I'm predicting this output
/*
line - all lines are once
global - this time just once
line - all lines are once
global - I should pop up 4x then 2x
global - I should pop up 4x then 2x
global - I should pop up 4x then 2x
global - I should pop up 4x then 2x
line - all lines are once
global - I should pop up 4x then 2x
global - I should pop up 4x then 2x
*/
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

[[a-k\]] line {
        print "line - all lines are once\n";
}

[[a-b\]] global {
        print "global - anything a through b\n";
}

//I'm predicting this output
/*
line - all lines are once
global - anything a through b
global - anything a through b
global - anything a through b
global - anything a through b
global - anything a through b
line - all lines are once
global - anything a through b
global - anything a through b
*/
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

[[^a-k\]] line {
        print "line - all lines are once\n";
}

[[^a-z\]] global {
        // now the sixth print the first time around might or might not
        // be correct depending on whether or not we count the newline
        print "global - anything but lower case letters: " + @match.length + " \""+@match
+"\"\n";

}
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

[[a-kA-K\]] line {
        print "line - all lines are once\n";
}

[[a-bM-Z\]] global {
        print "global - multiple ranges\n";
}

//I'm predicting this output
/*
line - all lines are once
global - multiple ranges
global - multiple ranges
global - multiple ranges
global - multiple ranges
global - multiple ranges
global - multiple ranges
line - all lines are once
global - multiple ranges
global - multiple ranges
global - multiple ranges
*/
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

// hopefully this is the correct way to say "space" in regexp
[ ] line {
        print "line - all lines are once\n";
}

[ ] global {
        print "global - for however many spaces\n";
}

//I'm predicting this output
/*
line - all lines are once
global - for however many spaces
global - for however many spaces
global - for however many spaces
global - for however many spaces
line - all lines are once
global - for however many spaces
global - for however many spaces
global - for however many spaces
global - for however many spaces
global - for however many spaces
*/
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

[\n] line {
        print "line - all lines are once\n";
}

[\n] global {
        print "global - the newline\n";
}

//I'm predicting this output
/*
line - all lines are once
global - the newline
*/
```

```
/*
Use input from file Mary.txt
regexp testing - ie: make sure the matching works and that
for global it repeats and for line it doesn't
*/

[\.] line {
        print "line - all lines are once\n";
}

[\.] global {
        print "global - periods\n";
}

//I'm predicting this output
/*
line - all lines are once
global - periods
*/
```

```
/*
last one to test on the basics list: replace
again on all used on the Mary.txt file
since it's basically the same format as insert, this will be very similar
only the output will be different
*/

[i] line {
        set $test, "this is a sentance\n";
        replace $test, 4, "REPLACE!";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 5;
        replace $testing, #integer, "replace";
        print $testing;
}

//I'm predicting this output
/*
thisREPLACE!ntance
thisREPLACE!ntance
testireplaceoperators
*/
```

```
/*
not gonna do as many simple ones because that is kinda pointless
i doubt stuff that works for insert won't work for this considering
how the format is the same
*/

[i] line {
        set $test, "this is a sentance\n";
        set $str, "REP";
        replace $test, 4, $str + "LACE!";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 2;
        replace $testing, #integer*2+1, "replace";
        print $testing;
}

//I'm predicting this output
/*
thisREPLACE!ntance
thisREPLACE!ntance
testireplaceoperators
*/
```

```
/*
not gonna do as many simple ones because that is kinda pointless
i doubt stuff that works for insert won't work for this considering
how the format is the same
*/

[i] line {
        set $test, "this is a sentance\n";
        set $str, "REP";
        set $str2, "CE!";
        replace $test, 4, $str + "LA" + $str2;
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 2;
        set $replace, "replace";
        replace $testing, 4+#integer-1, $replace;
        print $testing;
}

//I'm predicting this output
/*
thisREPLACE!ntance
thisREPLACE!ntance
testireplaceoperators
*/
```

```
/*
not gonna do as many simple ones because that is kinda pointless
i doubt stuff that works for insert won't work for this considering
how the format is the same
*/

[i] line {
        set $test, "this is a sentance\n";
        set $str, "R";
        set $str2, "CE!";
        replace $test, 3+1, $str+ "EP" + "LA" + $str2;
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 2;
        set $replace, "replace";
        replace $testing, 4+#integer-1, $replace;
        print $testing;
}

//I'm predicting this output
/*
thisREPLACE!ntance
thisREPLACE!ntance
testireplaceoperators
*/
```

```
/*
not gonna do as many simple ones because that is kinda pointless
i doubt stuff that works for insert won't work for this considering
how the format is the same
*/

[i] line {
        set $test, "this is a sentance\n";
        set $str, "R";
        set $str2, "CE";
        set #int, 4;
        replace $test, #int, $str+ "EP" + "LA" + $str2 + "!";
        print $test;
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 2;
        replace $testing, 4+#integer-1, "r" + "e" + "p" + "l" + "a" + "c" + "e";
        print $testing;
}

//I'm predicting this output
/*
thisREPLACE!ntance
thisREPLACE!ntance
testireplaceoperators
*/
```

```
/*
not gonna do as many simple ones because that is kinda pointless
i doubt stuff that works for insert won't work for this considering
how the format is the same
*/

[i] line {
        set $test, "this is a sentance\n";
        replace $test, 5, "replace past the final index";
        print $test + "\n";
}

[snow] line {
        set $testing, "testing for operators\n";
        set #integer, 2;
        replace $testing, 4+#integer-1, "r" + "e" + "p" + "l" + "a" + "c" + "e";
        print $testing;
}

//I'm predicting this output
/*
this replace past the final index
this replace past the final index
testireplaceoperators
*/
```

```
/*
testing basic sets for string variable with only strings and variables
this test and all the tests in this folder use the input found in Pangrams.txt
*/

[a] line {
        set $test, "one";
        set $test2, "one" + " plus ";
        set $test3, $test + " equals ";
        set $test4, $test2 + $test3;
        print $test4;
}

[b] line {
        set $test5, "four";
        print $test5 + "\n";
}

/*
one plus one equals one plus one equals four
one plus one equals four
one plus one equals four
one plus one equals four
*/
```

```
/*
hmm checking precedence, gotta look over docs before trying to figure out what the
actual output would be for /a/
*/

[a] line {
        set #test, 2*3;
        set $test2, 2 + 3 + 4;
        set $test3, 2 + 3 + #test + 4;
        print $test2 + " " + $test3 + "\n";
}

[b] line {
        set #test5, 4+4+4-6*2/3;
        print "checking operators: " + #test5 + "\n";
}

//I'm predicting this output
/*
9 15
9 15
checking operators: 8
9 15
checking operators: 8
9 15
checking operators: 8
9 15
checking operators: 8
*/
```

```
/*
if this computes the numerical value then changes it to string, it does what i think
otherwise it just might give a funny output.
Also, the ones before this test should compute first if this does compute first.
*/

[a] line {
        set #test, 2*3;
        set $test2, 2 + 3 - 4;
        set $test3, 2 - 3 + #test * 4;
        print $test2 + " " + $test3 + "\n";
}

[b] line {
        set #test5, (4+4+4)-(6*2)/3;
        print "checking operators: " + #test5 + "\n";
}

//I'm predicting this output
/*
1 23
1 23
checking operators: 8
1 23
checking operators: 8
1 23
checking operators: 8
1 23
checking operators: 8
*/
```

```
/*
now testing set with single character numbers and strings and variables
*/

[a] line {
        set $test, "1";
        set $test2, "1" + " plus ";
        set $test3, $test + " equals ";
        set $test4, $test2 + $test3;
        print $test4;
}

[b] line {
        set $test5, "4";
        print $test5 + "\n";
}

/*
1 plus 1 equals 1 plus 1 equals 4
1 plus 1 equals 4
1 plus 1 equals 4
1 plus 1 equals 4
*/
```

```
/*
this should be the same output as 2 BUT! we are testing for the automatic
conversion of numbers to strings
*/

[a] line {
        set $test, 1;
        set $test2, 1 + " plus ";
        set $test3, $test + " equals ";
        set $test4, $test2 + $test3;
        print $test4;
}

[b] line {
        set $test5, 4;
        print $test5 + "\n";
}

/*
1 plus 1 equals 1 plus 1 equals 4
1 plus 1 equals 4
1 plus 1 equals 4
1 plus 1 equals 4
*/
```

```
/*
mixing things up with concats of automatic convert number, variable, and string
and using set on that
*/

[a] line {
        set $test, "one";
        set $test2, " plus " + $test;
        set $test3, 1 + $test2 + " equals ";
        print $test3;
}

[b] line {
        set $test5, "4 four!!!";
        print $test5 + "\n";
}

/*
1 plus one equals 1 plus one equals 4 four!!!
1 plus one equals 4 four!!!
1 plus one equals 4 four!!!
1 plus one equals 4 four!!!
*/
```

```
/*
slowing moving over to simply testing setting number variables
*/

[a] line {
        set #test, 1;
        set $test2, " plus " + #test;
        set $test3, 1 + $test2 + " equals ";
        print $test3;
}

[b] line {
        set #test5, 4;
        print #test5 + "\n";
}

/*
1 plus 1 equals 1 plus 1 equals 4
1 plus 1 equals 4
1 plus 1 equals 4
1 plus 1 equals 4
*/
```

```
/*
more testing of combinations with sets
also starting to test the different operators
*/

[a] line {
        set #test, 1;
        set #test2, 1 + #test;
        set $test3, #test + " plus " + #test + " = ";
        print $test3 + #test2 + "\n";
}

[b] line {
        set #test5, 4*10;
        print "checking operators: " + #test5 + "\n";
}

/*
1 plus 1 = 2
1 plus 1 = 2
checking operators: 40
1 plus 1 = 2
checking operators: 40
1 plus 1 = 2
checking operators: 40
1 plus 1 = 2
checking operators: 40
*/
```

```
/*
more testing of combinations with sets
also starting to test the different operators
*/

[a] line {
        set #test, 2;
        set #test2, #test + 1;
        set $test3, #test + " plus " + #test + " = ";
        print $test3 + #test2 + "\n";
}

[b] line {
        set #test5, 4/2;
        print "checking operators: " + #test5 + "\n";
}

/*
2 plus 2 = 3
2 plus 2 = 3
checking operators: 2
2 plus 2 = 3
checking operators: 2
2 plus 2 = 3
checking operators: 2
2 plus 2 = 3
checking operators: 2
*/
```

```
/*
more testing of combinations with sets
also starting to test the different operators
*/

[a] line {
        set #test, 2;
        set #test2, #test + #test;
        set $test3, #test + " plus " + #test + " = ";
        print $test3 + #test2 + "\n";
}

[b] line {
        set #test5, 4-4;
        print "checking operators: " + #test5 + "\n";
}

/*
2 plus 2 = 4
2 plus 2 = 4
checking operators: 0
2 plus 2 = 4
checking operators: 0
2 plus 2 = 4
checking operators: 0
2 plus 2 = 4
checking operators: 0
*/
```

```
/*
hmm checking precedence, gotta look over docs before trying to figure out what the
actual output would be for /a/
*/

[a] line {
        set #test, 2+3;
        set $test2, #test + 10;
        print #test + " " + $test2 + "\n";
}

[b] line {
        set #test5, 4+4+4;
        print "checking operators: " + #test5 + "\n";
}

//I'm predicting this output
/*
5 15
5 15
checking operators: 12
5 15
checking operators: 12
5 15
checking operators: 12
5 15
checking operators: 12
*/
```

```
/*
if tests, all will be done on the input from file Mary.txt
*/

[Mary] line {
        print $substr("1234", 1, 3);
}
```

```
/*
while tests, all will be done on the input from file Mary.txt
*/

[Mary] line {

        print "print a 4 times: ";
        set #test2, 4;
        while(#test2 > 0){
                print "a";
                set #test2, #test2 - 1;
        }
        print "\n";

        print "print a 0 times: ";
        set #test2, 4;
        while(#test2 < 0){
                print "a";
                set #test2, #test2 - 1;
        }
        print "\n";

        print "print a 3 times (uses break): ";
        set #test2, 4;
        while(#test2 > 0){
                print "a";
                set #test2, #test2 - 1;
                if(#test2 == 1) {
                        break;
                }
        }
        print "\n";

        print "print a 4x4 matrix: \n";
        set #i, 4;
        while(#i > 0){
                set #j, 4;
                while(#j > 0){
                        print "a";
                        set #j, #j - 1;
                }
                print "\n";
                set #i, #i - 1;
        }


        print "print a 4x3 matrix using break: \n";
        set #i, 4;
        while(#i > 0){
                set #j, 4;
                while(#j > 0){
                        print "a";
                        set #j, #j - 1;
                        if(#j == 1){
                                break;
                                print "\nNObreak fails.\n";
                        }
                }
                print "\n";
                set #i, #i - 1;
        }
        print "\n";

        break; /* what happens here?*/

        print "break doesn't work for regexblocks.\n";


}
```

```
/*
Use input from file Mary.txt
while tests just have to test to see if the loop works
and that it works with all the different conditionals
*/

[Mary] line {
        set #count, 0;
        // this is like a for loop
        while (#count <= 3)
        {
                set #count, #count+1;
                print #count + "\n";
        }
}

[snow] line {
        set #counter, 3;
        // this is like a for loop
        while (#counter >= 0)
        {
                set #counter, #counter-1;
                print #counter + "\n";
        }
}

//I'm predicting this output
/*
1
2
3
4
2
1
0
-1
*/
```

```
/*
Use input from file Mary.txt
while tests just have to test to see if the loop works
and that it works with all the different conditionals
*/

[Mary] line {
        set #count, 3;
        // this is like a for loop
        while (#count == 3)
        {
                set #count, #count+1;
                print #count + "\n";
        }
}

[snow] line {
        set #counter, 3;
        // this is like a for loop
        while (#counter != 0)
        {
                set #counter, #counter-1;
                print #counter + "\n";
        }
}

//I'm predicting this output
/*
4
2
1
0
*/
```

```
/*
Use input from file Mary.txt
while tests just have to test to see if the loop works
and that it works with all the different conditionals
*/

[Mary] line {
        set #count, 0;
        while (#count < 3 && #count != 1)
        {
                set #count, #count+1;
                print #count +  "\n";
        }
}

[snow] line {
        set #counter, 3;
        while (#counter > 0 || #counter == 0)
        {
                set #counter, #counter-1;
                print #counter + "\n";
        }
}

//I'm predicting this output
/*
1
2
1
0
-1
*/
```

```
/*
Use input from file Mary.txt
while tests just have to test to see if the loop works
and that it works with all the different conditionals
*/

[Mary] line {
        set #count, 0;
        //this loop should not be entered
        while (#count)
        {
                set #count, #count+1;
                print #count +  "\n";
        }
}

[snow] line {
        set #counter, 3;
        // this loop should be entered
        while (#counter)
        {
                set #counter, #counter-1;
                print #counter + "\n";
        }
}

//I'm predicting this output
/*
2
1
0
*/
```