Final Project Report
For
The GG Programming Language

By
Kierstan Bell
Elizabeth Mutter
Jake Porway
and
Jonah Tower

Columbia University
COMS 4115 – Programming
Languages and Translators
Prof. Stephen Edwards
May 13, 2003

Table of Contents

1.	Introduction: White Paper	Page 3
2.	Language Tutorial	Page 6
3.	Language Reference Manual	Page 12
4.	Project Plan	Page 30
5.	Architectural Design	Page 35
6.	Test Plan	Page 38
7.	Lessons Learned	Page 39

APPENDIX:

A.	Complete Listing	Page 41
B.	Meeting Minutes	Page 94

The GG Programming Language White Paper

Kierstan Bell (klb2004@columbia.edu)
Elizabeth Mutter (eam2003@columbia.edu)
Jacob Porway (jmp204@columbia.edu)
Jonah Tower (jpt2002@columbia.edu)

Introduction

We live in a networking computer world where it is nearly a necessity to be able to develop network applications on a regular basis. However, current programming languages do not provide an easy and simple way with which to quickly develop such applications. The GG Programming Language is a simple programming language designed to minimize the work associated with creating network applications, and thus allow for easy and rapid prototyping and development of networking software.

Imagine that you are a software engineering student trying to complete a project for your semester's grade. As you become entangled in the tedious details of managing the many different networking components of standard programming languages, you are quickly running out of time. The *all-nighters* start to pile up on end and your ability to read the code on your screen diminishes. It becomes even harder and harder to understand the many error messages. However, if there was a language that provided very simple and easy to use interfaces to the most common networking standards and protocols, there would be no reason for you to face these difficulties. This is precisely where the GG programming language comes into play.

Body

General Technical Overview

At its top level the GG programming language provides programmers with several different networking functions that are simple and easy to use, but provide the essential building blocks for a variety of networking applications. In addition to these functions the GG programming language includes essential data types and flow control needed in order to develop basic working applications.

The basic syntax of the GG programming language finds its roots in languages like C and Java, thus making it easy for programmers experienced in those languages to learn the GG programming language. One of the largest motivators for the GG programming language is simply to provide the quickest and easiest way to get networking applications running, so having a structure that is not unlike that of Java or C helps programmers quickly get into coding and prevents bogging them down with learning complicated new syntaxes.

The GG programming language will translate to Java code using the GG programming language translator (GG-PLT). Then, using existing Java compilers, the Java code will be compiled to runnable Java classes. Due to the fact that Java runs on all platforms, this will give the GG programming language the same portability that Java already has. Also, this will allow for extensive future development of the language due to the sheer size and functionality of Java.

Networking

For the networking portion of the GG programming language the following approach is being taken: the language will provide programmers with several functions that make using TCP/IP socket connections simple and easy. Using the functions provided by the language each socket connection can be established with one or two lines of code and then transmission can occur immediately thereafter. To make a connection to a remote socket all that is needed if the hostname and port number of the remote system, then Strings of characters and even the entire contents of files can be transmitted with only a single function call.

```
void main(string arg1)
{
     file f = arg1;
     int port = socketcreate();
     socketconnect(port);
     send(f);
}
```

The above example will take the name of a file as an argument from the command line and then opens a socket for a client to connect to and then transmits the file's contents to the connecting client.

Syntax

To make the transition to the GG programming language easy for the many C, C++ and Java programmers, the syntax of the language will be much like that of C or Java. To continue the emphasis on creating a language that is quick and easy even in developing complex networking applications, the GG programming language will contain only the essential data types and flow control. Still, the syntax will look very much like that of Java.

```
int giveMeACoolInt()
{
     int cool;
     cool = 3;
     return cool;
}
```

Above is a small code snippet demonstrating how a function is defined.

Compiles to Java

Once an application is ready to be compiled and tested, the GG programming language translator (GG-PLT) will translate the GG code into Java and output a Java file. In addition to this, the GG-PLT will call the Java compiler and produce a runnable Java program that can then be used. This has a two-fold advantage: (1) by producing and making available the Java code for the GG applications, the GG-PLT allows programmers the flexibility to do more with their applications than what the current version of the GG programming language allows; and (2) by using Java as a basis for the language, GG applications are completely platform-transportable. This approach to developing applications will allow programmers to quickly prototype and then develop network applications using the GG programming language, but as noted above, they are still not limited by the constraints of the language.

Conclusion

GG is a compact language that enables programmers to quickly and efficiently design network applications. The network functions can easily be implemented without the characteristic nuisances associated with network programming, making for a robust, streamlined approach to creating connected applications. The output of Java applications as well as the Java source code allows for flexible cross-platform implementation and provides a quick way for the ambitious programmer to expand GG's functionality. GG takes the burden of network programming off of software designers and affords them the freedom they need to stop worrying and start creating.

Tutorial

Veteran programmers may find that the GG Programming Language syntax is similar to C++ or Java. In fact, much of the syntax of GG's control flow and variable manipulation are modeled closely after the Java language. For that reason, this tutorial assumes that the reader has a basic knowledge of programming in Java or C++ and jumps straight into introducing you to the more interesting aspects of GG. Most people with experience programming should have no problem picking up GG's familiar style, but if you're feeling a bit rusty on your Java, feel free to flip to the Appendix at the end of the tutorial for details on the more mundane aspects of GG's control flow. So, without further ado, let's write our first GG program.

1 - Hello, World!, GG Style

GG was originally designed to eliminate the hassles of network programming, so let's make a simple client/server application to get familiar with how GG manipulates sockets. Start by typing the following into a plain text file document:

```
//This is our server

void main()
{
   int port = socketcreate();
   string hostname = getlocalhost();

   print("Hostname: ");
   print(hostname);
   print("\nPort: ");
   print(port);

      socketaccept(port);


   int counter = 0;

   while(counter < 1000)
   {
      send(itos(getTime()));
      counter++;
   }

   send("Done");

   socketclose(port);
}
```

Let's take a look at what we wrote before we move on to the client:

`1: void main()` - This just tells our program that this is where it should start. This function can also take arguments from the command-line at runtime, but we'll worry about that later.

`3: int port = socketcreate()` - This line creates an integer called port and assigns it to the result of the function socketcreate(). socketcreate() simply grabs the first empty port that it finds on your computer and returns it. We can now refer to the port that we've opened by using the port variable, for example when we close it on the last line.

`4: string hostname = getlocalhost()` - getlocalhost() returns the name of the current host. We may need this later, for example to let our clients know what host to connect to.

`7: print(hostname)` - Every good language has a print statement and, appropriately, GG's is just "print". This will print our hostname to the standard output.

11: socketaccept(port) - The socketaccept(int port) function blocks until a client connects to the socket that matches the port number given as an argument. Once the socket has been connected messages can be passed through the socket.

`17: send(`itos(getTime`()))` - Here we're passing another function as an argument to a function. Let's work from the outside in: send() is a function that takes a string and sends it through to a client. "But what if there are many clients? Which one receives the message?", you may ask. Well, send can actually takes one or two arguments. The second argument specifies which port you would like to send the string to but, if no port is specified, all currently connected clients receive the message. We'll look at this feature in more depth later on. Now, we come to the itos() function, which takes an int as an argument and then returns it in string form. Moving along, getTime() is a function that returns the number of second since February 09, 2003 at 9:00pm EST. Now you can see that this line sends the current time as a string to all the clients connected.

`23: socketclose(port)` - Now that we're done sending the time, let's close our socket. This just makes things tidy before our program exits.

Don't worry if you don't understand everything about what this program does yet. Hopefully it will become clearer we actually write our client, which we'll do now. Start up a new plain text file and enter the following:

```
void main()

{
    print("Enter hostname: ");
    string hostname = getline();
    print("Enter port number: ");
    int port = stoi(getline());
    port = socketconnect(hostname, port);

    string returnstring = recv();
    while(returnstring != "Done")
    {
         print(returnstring);
         returnstring = recv();
```

```
    }

   socketclose(port);
}
```

So let's take a look at how the client uses GG's network functions.

`4: string hostname = getline()` - getline() is GG's way of getting user input from standard in. This line simply means that whatever the user types onto the screen will be stored in the variable hostname.

`6: int port = stoi(getline())` - Here we're using getline() again, but its result is first passed to stoi(). stoi() stands for String TO Integer and it, well, turns a string into an integer. We can then read in the port number from the command line and store it in the port variable. This could have also been done with int port = getint() which simply gets the first int that the user enters into the command line.

`7: port = socketc`onnect`(hostname, port)` - Notice that this time we call socketconnect with two variables, the hostname and the port. In our server, we simply wanted a socket that would sit on the current computer. In our client, however, we now want to create a socket that connects to a specific port on another computer.

`9: string returnstring = recv()` - The counterpart so send(), recv() receives a string from the socket we created. As in send, we could specify a port to receive from, but we only have one open, so recv() will suffice. If there were multiple ports open, recv() would look to find something from the lowest connected port number.

Now, just compile these two files with the GG compiler. Run server first, followed by client. You should see the time print out for a couple of seconds, after which the socket is closed. Congratulations! You've just written your first GG program (programs in fact...).

2 - File Example

Believe it or not, we've just seen almost all of the basic functionality of GG's networking classes. GG's other big hook is the ability to manipulate files. Files are represented in GG by the file data type, which represents a file on your computer. Let's walk through an example:

```
void main(string arg1, string arg2)

{
   file file1 = arg1;
   file file2 = arg2;

   String carryover;

   carryover = fgetline(file1);

   while(carryover != EOF)
   {
      fprint(file2, "append", carryover);
      carryover = fgetline(file1);

   }
}
```

So, let's break this program down and see what it does.

1: `void main(string arg1, string arg2)` - Here we have the main method declared with two arguments included in it. This then allows the user to enter input into the program directly from the call to initialize the program. This is done as in C or Java by placing the arguments to be read directly after the call to the program in a space-separated list.

3: `file file1 = arg1` - Here we see how to use the file data type that is a part of the GG programming language. By simply setting the variable for the file data type equal to a string that contains the name of the file the correct file will be associated with that variable name.

8: `carryover = fgetline(file1)` - To get a line of text from the file represented by the variable name file1 we simply need to make a call to the fgetline() function and include file1 as the argument to the function. fgetline() will continue to get the next line of the file until it reaches the end, where the next call to fgetline() will then get the first line of the file. GG will keep track of where in each file one currently is, so that one can interleave calls to fgetline() to different files and not have to start over each time.

12: `fprint(file2, "a`append", carryover) - To then print a line of text to a file we use fprint(), which works exactly like print() but it also takes a file as an argument and then the second argument is either the string "append" or "overwrite" indicating how the line of text should be written to the file.

And, that, in a nutshell, is a quick and simple GG program to copy one file to another. Notice that it took less than 10 lines of real code, and that it works very simply and cleanly.

In these past two we have covered nearly all there is to know about the special functions of the GG programming languages. However, this simplicity combined with a bit of ingenuity can allow a crafty programmer to create a vast number of networking applications in a very short time span and without having to worry about a lot of the nitty-gritty that C requires or the complex classes and method calls of Java.

3 - VERY basic review of control flow and assignments

So, let's now do a really quick review of the basics of control flow and variable declaration and value assignment. First to declare a variable of any of GG's data types the format is as follows:

<data type> <variable name>;

Then to assign a value to that variable you just do the following:

<variable name> = <value>;

Pretty simple, huh? You can also combine the two statements:

<data type> <variable name> = <value>;

How cool.

The control flow in GG are handled with two forms: the "if else" and the "while" loop. To use the control flow features of GG is a very simple. An if-else statement is done as follows:

```
if(<boolean>)
{
      <statements>
}
else
{
      <statements>
}
```

It is important to remember that the braces at the beginning and end of each block of statements are required even if there is only one line of cod there. The <boolean> can be either a variable that is a boolean or it can be a comparison done with one of the built in comparison operators. And, the while loop looks as follows:

```
while(<boolean>)
{
      <statements>
}
```

So, now you know all there is to the basics of GG; go out to fill the world with endless programs using this marvelous language.

The GG
Reference Manual

Kierstan Bell
Elizabeth Mutter
Jacob Porway
Jonah Tower
Columbia University
New York, New York 10027

Table of Contents

THE GG ..... **I**

REFERENCE MANUAL ..... **I**

KIERSTAN BELL ..... **I**

TABLE OF CONTENTS ..... **II**

..... **II**

1. INTRODUCTION ..... **- 1 -**

2. LEXICAL CONVENTIONS ..... **- 1 -**

2.1 COMMENTS ..... - 1 -
2.2 IDENTIFIERS ..... - 1 -
2.3 RESERVED WORDS ..... - 1 -
2.4 TYPES ..... - 2 -
2.4.1 Boolean (boolean) ..... *- 2 -*
2.4.2 Character (char) ..... *- 2 -*
2.4.3 Filename (file) ..... *- 3 -*
2.4.4 Integer (int) ..... *- 3 -*
2.4.5 String ..... *- 3 -*

3. EXPRESSIONS ..... **- 4 -**

3.1 PRIMARY EXPRESSIONS ..... - 4 -
3.1.1 identifier ..... *- 4 -*
3.1.2 (expression) ..... *- 4 -*
3.2 UNARY OPERATORS ..... - 4 -
3.2.1 - expression ..... *- 4 -*
3.2.2 ! expression ..... *- 4 -*
3.3 MULTIPLICATIVE OPERATORS ..... - 4 -
*3.3.1 expression * expression* ..... *- 5 -*
3.3.2 expression / expression ..... *- 5 -*
3.4 ADDITIVE OPERATORS ..... - 5 -
3.4.1 expression + expression ..... *- 5 -*
3.4.2 expression - expression ..... *- 5 -*
3.5 RELATIONAL OPERATORS ..... - 5 -
3.6 EQUALITY OPERATORS ..... - 6 -
3.6.1 expression == expression ..... *- 6 -*
3.6.2 expression != expression. ..... *- 6 -*
3.7 LOGICAL OPERATORS ..... - 6 -
3.7.1 EXPRESSION && EXPRESSION ..... - 6 -
3.7.2 EXPRESSION || EXPRESSION ..... - 6 -
3.8 ASSIGNMENT OPERATORS ..... - 6 -
3.8.1 leftop = expression ..... *- 7 -*

4. DECLARATIONS ..... **- 7 -**

4.1 TYPE SPECIFIERS ..... - 7 -

TABLE OF CONTENTS

4.2 DECLARATORS ..... - 7 -

5. STATEMENTS ..... - 7 -

5.1 EXPRESSION STATEMENT ..... - 8 -
5.2 IF STATEMENT ..... - 8 -
5.3 WHILE STATEMENT ..... - 8 -
5.4 RETURN STATEMENT ..... - 8 -

6. SCOPE RULES ..... - 8 -

7. FUNCTIONS ..... - 9 -

7.1 FUNCTION DECLARATIONS ..... - 9 -
7.1.1 Parameters ..... - 10 -
7.1.2 Function Body ..... - 10 -

Preface

The GG programming language was designed by Kierstan Bell, Elizabeth Mutter, Jacob Porway and Jonah Tower to minimize the work associated with creating Socket and FTP network applications, and thus allow for easy and rapid prototyping and development of these applications. This manual is a specification of the syntax and semantics of the language. Although this is a thorough, workable language, it is a simple language, which will allow for the evolution of this language.

1. Introduction

This manual describes the GG programming language. Some of the standards below are modeled after those stated in the C and Java Reference Manuals as noted. GG is a compact language that enables programmers to quickly and efficiently design Socket and FTP network applications. The functions supplied by the "Network Library" can easily be implemented without the characteristic nuisances associated with network programming, making for a robust, streamlined approach to creating connected applications. The output of efficient Java applications as well as the Java source code allows for flexible cross-platform implementation and provides a quick way for the ambitious programmer to expand GG's functionality.

2. Lexical Conventions

There are several types of tokens: identifiers, reserved words, types, expression operators, and other separators. White space is ignored and is required to separate tokens. White space is defined to be the ACII space, horizontal tab and line terminators. Comments are also ignored, and can be used to separate tokens.

2.1 Comments

Same as the Java programming language, the characters / * introduce a multiple line comment, which terminates with the characters * /. The characters // introduce a single line comment, which terminates with the line terminator.

```
//This is a single-line comment
/* This is also a comment,
   a multiple-line comment */
```

2.2 Identifiers

As noted in the C Reference Manual, an identifier is a sequence of letters and digits; the first character must be alphabetic. The underscore "_" counts as alphabetic. Identifiers are case sensitive.

2.3 Reserved words

The following identifiers are reserved for use as keywords, and may not be used otherwise:

append	boolean
cd	char
else	false
file	if

int	ls
mkdir	overwrite
put	return
rm	rmdir
string	threaded
true	void
while	

abstract	break
byte	case
catch	class
const	continue
default	do
double	extends
final	fainally
float	for
goto	implements
import	instanceof
interface	long
native	new
package	private
protected	public
return	short
static	staticfp
super	switch
synchronized	this
throw	throws
transient	try
volatile	

2.4 Types

There are several types, Boolean (boolean), Character (char), Filename (file), Integer (int) and String (string), which are as follows,

2.4.1 Boolean (boolean)

As in the Java programming language, An identifier of type boolean has exactly two values: true and false, which are indicated by the Integer values 1 and 0, respectively.

2.4.2 Character (char)

Much like how character constants are noted in the C Reference Manual, an identifier of type char is expressed as a character or an escape sequence, enclosed in ASCII single quotes, whose values are 16-bit unsigned integers representing Unicode

characters. A single quote must be preceded by a backslash " \ " and certain non-graphic characters, and " \ " itself, may be escaped according to the following table:

BS	\b
NL	\n
CR	\r
HT	\t
ddd	*ddd*
\	\\

A special case of the construction "*dd*d" is "\0", which indicates a null character.

2.4.3 Filename (file)

An identifier of type file is expressed as a type String, which represents the filename. The following is an example of declaring an identifier of type file,

file myFile1 = "foo.txt";
file myFile2 = "/home/foo.txt"

If the file "foo.txt" does not exist, it is created in the specified directory.

2.4.4 Integer (int)

As noted in the Java Reference Manual, an identifier of type int is a sequence of one or more digits whose value is a 32-bit two's complement integer.

2.4.5 String

As described in the C Reference Manual, an identifier of type String is a sequence of characters surrounded by double quotes. A string has the type array-of-characters (see below) and refers to an area of storage initialized with the given characters. The compiler places a null byte (\0) at the end of each string so that programs which scan the string can find its end. In a string, the character " " " must be preceded by a " \ " ; in addition, the same escapes as described for characters may be used. Strings take the following form, where *StringCharacters* are optional,

String:
    "StringCharacters"
StringCharacters:
    StringCharacter
    StringCharacters StringCharacter
StringCharacter:
    InputCharacter but not " or \
    EscapeSequence

A line terminator can not appear within the double quotes.

The following are examples of string literals:

```
"  "                    //the empty string
" \" "                  //a string containing " alone
"This is a string"      // a string containing 16 characters
```

3. Expressions

Same as the layout of the C reference manual, the precedence of expression operators is the same as the order of the major subsections of this section (highest precedence first). Within each subsection, the operators have the same precedence.

3.1 Primary expressions

3.1.1 identifier

An identifier is a primary expression. Its type is specified by its declaration.

3.1.2 (expression)

A parenthesized expression is a primary expression. Its type and value are identical to those of the bare expression. Parentheses may be used to specify the order of evaluation.

3.2 Unary operators

Expressions with unary operators group right-to-left and are similar to those in C.

3.2.1 - expression

The result is the negative of the expression, and has the same type. The type of the expression must be int.

3.2.2 ! expression

The result of the logical negation operator ! is 1 if the value of the expression is 0, 0 if the value of the expression is non-zero. The type of the result is boolean. This operator is applicable only types to boolean and int.

3.3 Multiplicative operators

The multiplicative operators * and / group left-to-right. They return the product and quotient respectively of two values.

3.3.1 expression * expression

The binary * operator indicates multiplication. Expression and the variable the result is assigned to must be of type int.

3.3.2 expression / expression

The binary * operator indicates division. Expression and the variable the result is assigned to must be of type int.

3.4 Additive operators

The additive operators + and - group left-to-right. They return the sum and difference of two operands respectively.

3.4.1 expression + expression

The result is the sum of the expressions. Expression and the resulting variable must be of type int. the operation performed is dependent on the type of the resulting variable. the resulting variable is assigned the sum of the expression. If the resulting variable is of type string, the resulting variable is a reference to a newly created string that is the concatenation of the two operand strings. The characters of the left-hand operand precede the characters of the right-hand operand in the newly created string. If the resulting variable is of type file, the resulting value is a reference to a newly created file object that is the concatenation of the contents of the two operand files. The characters of the left-hand operand precede the characters of the right-hand operand.

3.4.2 expression - expression

The binary - operator indicates subtraction. Expression and the variable the result is assigned to must be of type int.

3.5 Relational operators

expression < *expression*
expression > *expression*
expression <= *expression*
expression >= *expression*

The relational operators group left-to-right and always return a boolean. The operands must be of type int and are compared based on their numeric values. The operators < (less than), > (greater than), <= (less than or equal to) and >= (greater than or equal to) all yield 0 if the specified relation is false and 1 if it is true.

3.6 Equality operators

The expressions must be of equal type. An expression may be of type string implementing a .compareTo(). The type of an equality expression is always boolean.

3.6.1 expression == expression

The operator == (equal to) yields true (1) if both of the operands are true, false (0) otherwise. If expression is of type char, int, float, or boolean, equality is determined by the numerical values of the expressions. If expression is of type string, equality is determined by the characters of each string. If expression is of type file, equality is determined by the contents of each file.

3.6.2 expression != expression

The operator != (not equal to) yields false (0) if both of the operands are true or both false, true (1) otherwise. If expression is of type char, int, float, or boolean, equality is determined by the numerical values of the expressions. If expression is of type string, equality is determined by the characters of each string. If expression is of type file, equality is determined by the contents of each file.

3.7 Logical Operators

The logical operators group left-to-right and always return a boolean. The expression must be a relational operator expression they can be ints where nonzero is true and zero is false, and is evaluated as described above in Section 3.6.

3.7.1 expression && expression

The operator && is evaluated from left-to-right. The && operator returns 1 if both its operands are non-zero, 0 otherwise. The first operand is evaluated first. If its value is false, then the operator && yields false and the second operand is not evaluated. If the value is true, then the second operand is evaluated and the operator && yields its value.

3.7.2 expression || *expression*

The operator || is evaluated from left-to-right. The || operator returns 1 if either of its operands is non-zero, 0 otherwise. The first operand is evaluated first. If its value is true, then the operator || yields true and the second operand is not evaluated. If the value is false, then the second operand is evaluated and the operator || yields its value.

3.8 Assignment operators

All of the assignment operators group left-to-right. All left operands are expressions referring to a region of storage that can be manipulated, and the type of an

assignment expression is that of its left operand. The value is the value stored in the left operand after the assignment has taken place.

3.8.1 leftop = expression

The value of the expression replaces that of the left operand. The operands must be of the same type.

4. Declarations

Pulling ideas from the C Reference Manual, declarations are used within function definitions to specify the interpretation given to each identifier; they do not necessarily reserve storage associated with the identifier. Declarations have the form

declaration:

type-specifier declarator-list;

The declarators in the declarator-list contain the identifiers being declared.

4.1 Type specifiers

The type-specifiers are all those discussed in section 2.4. The type-specifier cannot be missing from a declaration.

4.2 Declarators

The declarator-list appearing in a declaration is a comma-separated sequence of declarators.

declarator-list:

declarator

declarator , declarator-list

The specifiers in the declaration indicate the type of the objects to which the declarators refer. Declarators have the syntax:

declarator:

identifier

declarator ( ) //declares a function

declarator [const-expr] //declares an array

s(*declarator*) //does not change the type, changes binding

5. Statements

The statements below are similar to those descriptions found in the C and Java reference manuals.

5.1 Expression statement

Most statements are expression statements, which have the form

expression ;

5.2 If statement

The If statement can take the following two forms, where the *expression* is type boolean,

if (*expression*) *statement*
if (*expression*) *statement* else *statement*

In both of the above cases, if the expression is true the first statement is executed. In the first case, if the expression is false, no action is taken. In the second case, if the expression is false the second statement is executed.

5.3 While statement

The while statement takes the following form, where the *expression* is type boolean,

while (*expression*) *statement*

The statement is executed repeatedly until the value of the expression is false.

5.4 Return statement

The return statement can take the following two forms,

return ;
return (*expression*) ;

A function (see section 8 below) returns a value by means of the return statement. In the first case, no value is returned and this statement must be contained in a function that is declared not to return a value (using the keyword void). In the second case, the value of the *expression* is returned and this statement must be contained in a function that is declared to return a value that is of the same type that the function is declared to return.

6. Scope rules

A complete GG program must be contained in one file. The file must contain the main function, which takes 4 optional command line arguments of any primitive type, and returns any value (the return value must be specified as “void” if no value is to be returned). Two Java files will be created for the main function, which will be named . . . Global variables are declared at the top of a GG file, outside of a function and their scope extends from their definition through the end of the GG file. The scope of functions declared (see section 8.1) extends from their definition through the end of the GG file. The scope of local variables extends from their definition through the end of the function in which they are declared. All variables and functions must be defined before they are used, thus making the main function the last function declared in a GG file by default.

A function that is declared as being threaded, results in the generation of a separate Java file, which will be named fooThreadedMethod.java for a threaded function named “foo”.

7. Functions

All GG files must contain the main function. A Programmer may declare new functions and invoke them within the main function. t is not permitted to make statements out side of the scope of a function definition at all. A programmer may also invoke predefined functions from the “Networking Library” (see appendix) within the main function. The function invocation expression is as follows, where the *ArgumentList* is optional,

FunctionInvocation:
    FunctionName (*ArgumentList*)
ArgumentList:
    Argument
    Argument, ArgumentList
Argument:
    DataType VariableName

7.1 Function Declarations

Similar to how methods are described in the Java Reference Manual, in the GG programming language, a function declares executable code. Invoking a function (see above) will invoke the executable code. A function is declared as follows,

FunctionDeclaration:
    FunctionHeader FunctionBody
FunctionHeader:
    threaded *ResultType FunctionDeclarator*
ResultType:
    Types (see section 2.4)
    void
FunctionDeclarator:
    Identifer (*ParameterList*)

The *FunctionHeader* may include the optional keyword “threaded,” which will create a separate thread for this function. A function declaration either specifies the type of value that the function returns or uses the keyword void to indicate that the function does not return a value. The *Identifier* in a *FunctionDeclarator* is used to call the function. Two new functions may not have the same *FunctionHeader*, but they may be overloaded (have the same *Identifier*) because they can be distinguished by the *ParameterList*. But, the predefined functions may not be overloaded, therefore the *Identifier* may not be “main” or the same as any of the functions described in the appendix. Also, because GG is tanslated to Java before it is run, there can be no use of the Java reserved words list.

7.1.1 Parameters

The *parameters* of a function, if any, are specified by a list of comma-separated parameter specifiers. Each parameter specifier consists of a type and an identifier that specifies the name of the *parameter*:

ParameterList:
Parameter
ParameterList , Parameter
Parameter:
Type Identifier

If a function doesn't have parameters, then an empty pair of parentheses appears in the *FunctionDeclarator*. Two parameters in the same function cannot have the same *Identifier*. The *Identifier* may be used as a name in the body of the function to refer to the parameter and may not be re-declared within the function body. The scope of a parameter is the entire body of the function. There can be no overloading of identifiers in a completely general sense.

7.1.2 Function Body

A *function body* is a block of executable code. If the *ResultType* of the function is void, then its body must not contain a return statement that has an *expressio*n. If the *ResultType* of the function is one of the types discussed in section 2.4, then any return statement in the body must have an expression of the same type. The function body takes the following form, where the *declaration-list* is optional,

FunctionBody:
{ declaration-list statement-list }

Appendix

Function Library

The GG programming language provides a function library that offers general all-purpose functions for performing common programming tasks such as reading and writing to files or the command line. GG also provides within this library a collection of functions that perform common networking tasks for means of enabling programmers to quickly and efficiently design Socket and FTP network applications.. All of the following functions may be implemented on their own thread by writing a new wrapper function that is declared to be threaded in the *FunctionHeader*. For example,

```
threaded myFunc(int foo){
        while(1){
                print("This is a threaded print function in an infinite loop.");
        }
}
```

1. General Functions

1.1 fgetline(file)

This function takes 1 argument, a *file,* and returns a variable of any type (specified by its assignment, with auto-conversion if necessary, see section 3.1). This function reads the specified file line by line, starting at the beginning of the file, and returns the current line. Therefore, for example, if you would like to return the 5^{th} line of the specified file, this is accomplished as follows,

```
String FifthLine;
for (int i=0; 1<5; i++){
        FifthLine = fgetsome("myFile");
}
```

The final string value stored in *FifthLine* is the contents of the fifth line of *myFile*. Now, if you would like to parse this line of *myFile*, you could do so, for example, knowing that a string is an array-of-characters. Knowing this, you could loop through the array by incrementing the index into the array and parse by characters.

1.2 fprint(file, append | overwrite, variable)

This function takes 3 arguments, a *file*, *append* or *overwrite* keyword and a *variable* of any type. The third argument may be any concatenation of variables. This function prints the *variable* to the specified *file* at the beginning or end of the file, specified by the append and overwrite keywords, respectively.

1.3 print(variable)

This function takes 1 argument, a *variable* of any type. As described above, the third argument may be any concatenation of variables. The *print* function takes an argument of any type and prints it as a string to the terminal, with the exception of type filename. If the argument is type filename, the contents if the specified file is printed to the terminal. An example of the implementation of the *print* function is as follows, if char c = 5; has been declared, then

print(c); = print('5');

1.4 getline( )

This function takes no arguments and returns a variable of any type (specified by its assignment, with auto-conversion if necessary, see section 3.1). The *getsome* function reads from the command line and returns what has been entered on the command line. This function will first read in what has been entered then typecast.

1.5 getTime( )

This function takes no arguments and returns the number of seconds since February 09, 2003 at 9:00pm EST[1].

1.6 itos()

This function accepts a int value and will return a string containing that integer.

1.7 stoi()

This function accepts a string and attempts to convert it to an int and returns that value.

2. Network Initialization Functions

The following functions are provided for quick and easy initialization of server and client socket and ftp connections. Only one ftp connection is allowed per GG file.

[1] Date and tine the first meeting minutes were taken at a GG development group meeting

2.1 sockcreate(int port)

This function takes 1 optional argument, an int, and returns an int. This function creates a server socket by grabbing the next available port, or at the port number specified by the argument passed. It returns the port number for the socket it created or -1 upon failure. For example if this function is passed the port number 22, you then have an ftp server.

2.2 sockconnect(int port)

To set up the server socket to listen, open the server, this function takes 1 argument, an int, which specifies a port number (the value returned by the sockcreate function).

2.3 sockconnect(string host, int port)

To open a local port and connect to the server, this function takes 2 arguments, a string that specifies the server hostname and an int that specifies port number. This function returns the local port number or -1 upon failure.

3. Socket Functions

Once the server and client socket and ftp connections are made, the following functions become available.

3.1 send(“ ”)

This send function may be used once a socket connection has been initialized. This function takes two argument, a String to send across the socket (if the passed argument is not of type String, it is automatically converted, see section 3.1) and an optional int. The optional second argument specifies the port number to send across (since multiple socket connections may be initialized). If this argument is not specified, it defaults to the lowest port number.

3.2 recv( )

This recv function may be used once a socket connection has been initialized. This function takes one optional argument one optional argument of type int, and returns a variable of any type (specified by its assignment, with auto-conversion if necessary, see section 3.1). This function receives what is sent across the socket and returns that value. The optional argument specifies the port number to receive across (since multiple socket connections may be initialized). If this argument is not specified, it defaults to the lowest port number.

References

1. Ritchie, D. M. “C Reference Manual.” Bell Telephone Laboratories,
2. Gosling, J., Joy, B., Steele, G., Bracha, G. “The Java™ Language Specification, second edition.” http://java.sun.com/docs/books/jls/

Project Plan

Roles and responsibilities

The members of the GG team have learned that in programming a large, group software project, one is required to wear many hats. Despite our initial division of labor, many members were asked to help with parts of the project outside of their jurisdiction. While this may not have been the division originally desired, the new setup allowed each member of the team to become more familiar with the working of the language and the design process as a whole. The original roles and responsibilities of each team member are listed below.

Kierstan Bell: Documentation

Responsibilities include writing the white paper, language reference manual, and final report

Elizabeth Mutter: Front End

Responsibilities include writing a parser and Lexer in ANTLR for the GG language.

Jake Porway : Testing

Responsibilities include designing a test suite that thoroughly tests the functionality and semantic checking of the GG compiler and documenting the tests performed.

Jonah Tower : Back End

Responsibilities include writing a static semantic checker in ANTLR to check GG files and generation of Java code from the AST.

Process used for planning...

The GG Programming language was made possible through following a carefully thought out schedule. Biweekly meetings were arranged to ensure that the project moved at a reasonable pace throughout the semester. During the first half of the semester, these

meetings were dedicated to specifying the desired functionality of the language and determining how to implement said functionality. Although the entire group was present for these brainstorming meetings, individual roles began to take shape as each member was responsible for focusing on ideas for the implementation of various parts of the language, such as the back end. Deadlines were also set weekly so that each member knew what he/she was responsible for by the next meeting. In this manner, the language was clearly specified and a long-term project goal was established by the first half of the semester.
By mid-April, roles had been clearly defined and the meetings were used merely to check on each team member's progress as the actual compiler was being written, tested, and documented. Development and testing occurred nearly simultaneously as members met each week to integrate the parts of the language due for that week. The front end and back end were tested together to ensure that syntactically and semantically correct code was being output for each new feature added. Through this regimented process of setting weekly deadlines and meeting at least weekly for integration sessions, the project progressed successfully through the semester.

Programming style

At no point did the group really sit down and develop a programming style for us all to adhere to. Instead we divided up the project cleanly and gave each group member (or subgroup) the freedom to work as they felt necessary. And, then to make sure that the different parts of the project would work well together we were careful to define the interactions between each of the subcomponents. To facilitate this we had two weekly meetings and we setup an FTP account for the group so that information could easily be passed from group member to group member.

For the documentation we utilized the group FTP account in the following way. Each part of the documentation was assigned to a single group member, who then put together the rough draft. That group member would then be responsible for uploading that document to the correct location on the FTP server and then notifying the next group member that they should take a look at what was there. Then that group member would ass what they felt was necessary and continue the process by notifying the next group member. In this way each group member was able to give their own input to the documentation and by the time it went all the way around the document was complete. Any conflicts were then resolved by the group member responsible for the document. This was a very workable system, because our group was only four members.

The front end ANTLR coding was handled primarily by Liz and Kierstan using an extreme programming approach. Then, Jake became involved in that part of the project as well and they worked a sort of round-robin/extreme-programming system of working their way through the front end of the language.

The testing was primarily designed and implemented by Jake on his own; however, as he became more and more involved in the front end of the project the testing became more

intertwined with that part of the project as well and they were developed in parallel. Many of the programs in the test suite week written by a C++ program, which simply out put dozens of GG programs that tested the basic functionality

Finally, the backend was developed solely by Jonah. Or, that is that the methods being used he developed in Java. To implement the functions that GG provides the programmer we simple created a class called GG_guts in Java that had all the same function calls. We did the same thing for the file data type by creating a GG_file class. Providing these Java methods and the fact that the syntax of GG is very similar to that of Java this makes the translation to Java classes from GG code a very clean and simple task.

Planned project timeline

2.2.03 - Determine what GG will actually be, i.e. come up with a language
2.11.03 - Begin hashing out grammar, outlining test suite
2.16.03 - White paper complete, begin language reference manual

3.02.03 - Check-up: Development on Lexer/Parser, back end, reference manual, and test suite
3.12.03 - Rough draft of language reference manual complete
3.16.03 - Language reference manual completed
3.24.03 - Parser complete

4.01.03 - Code generation complete
4.17.03 - Static semantics complete

5.01.03 - Testing complete
5.13.03 - Project complete

Actual project timeline (Project log)

2.2.03 - The idea for the GG language is born: A GUI based networking language
2.11.03 - Basic functionality of GG is nailed down: Protocols and GUI functionality lay out
2.16.03 - White paper complete, begin language reference manual. Test suite also begins.
2.24.03 - GUI and network function syntax and semantics are defined.

3.5.03 - The GUI gets scrapped, on recommendation of Prof. Edwards. GG is now a networking language
3.10.03 - In an attempt to move away from functional languages, GG takes on a scripting feel with the idea of auto-conversion, i.e. automatic casting of variable types
3.20.03 - Rough draft of the language reference manual complete
3.26.03 - Language reference manual complete

4.6.03 - The entire group has a massive panic attack. Emergency meetings are called and auto-conversion goes the way of our GUI.
4.17.03 - As opposed to project plan, the parser and back end proceed in tandem, testing small chunks over time.

5.1.03 - It's May. Parser complete
5.5.03 - Code generation complete
5.8.03 - Static semantics complete
5.12.03 - Presentation complete
5.13.03 - Presentation time

Tools and languages used

Front end:

- ANTLR used to design Lexer, Parser, and Walkers for both code generation and semantic checking

Back end:

- JBuilder used for Java classes available through GG

Test suite:

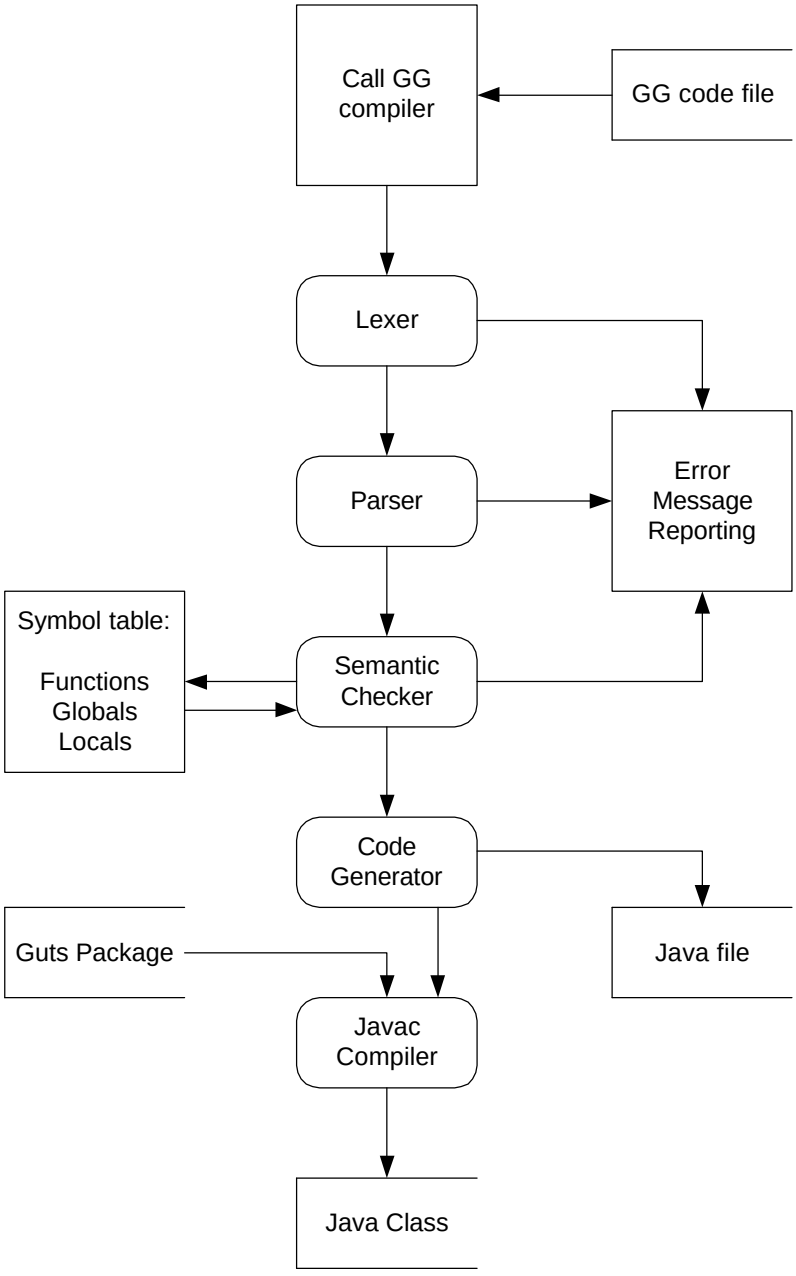
- C++ used to create batches of test files
- Emacs used to write specific files
- Shell to test

Documentation:

- Microsoft word for white paper and language reference manual
- The surprisingly capable KOffice for the final report.

Architectural Design

As described in the white paper, the GG compiler translates GG code into Java code, which is then compiled into the final GG program. As is common in most programming languages, this compiler consists of a Lexer, a Parser, a Semantic Checker with included symbol tables, a Code Generator, and the Runtime environment, which is Java. The basic compilation process is depicted below.



Compiler Start/The Lexer

As in most compilers, the process above begins with a GG file being fed into the compiler, which parses any command-line arguments and passes them along with the file to the Lexer for tokenization. If any combinations in the character stream aren't recognized by the Lexer, an "unknown character" error is thrown. The result of the Lexer, a stream of tokens, is then passed into the Parser.

The Parser

The Parser scans the input token stream, matching nodes with the grammar of the language and building an abstract syntax tree out of the nodes. If any syntactic combinations that are unrecognized by the Parser occur in the file, the Parser halts and returns an error to the main compiler. If no errors are found in the parser and all tokens are built successfully into a syntax tree, the Parser passes the tree onto the semantic checker.

The Semantic Checker

The semantic checker is invoked next to walk the tree and ensures semantic correctness. In order to deal with both problems of scope and overloaded functions, the Semantic Checker is given possession of three hash tables to keep track of declared and initialized variables. The first table, the Global table, is used to store all variables that are globally accessible in the program. The second table, the Local table, instead stores all of the variables that are declared in local scopes. Any time the walker enters a new scope block, which is designated by an open and a closed curly brace ("{ }"), the local table is cleared, and all variables for this block are created and checked within this table. If a variable is identified that does not exist in the Local table, the Semantic Checker also checks if it is instead declared in the Global table before throwing an error. For ease of semantic analysis, each of the two variable tables above contain not only the name and type of the variable, but also a flag specifying whether it has been initialized yet or not. This prevents the user from trying to use variable that have yet to be assigned a variable. The last table, the Function table, contains a list of all functions that are available. In GG, all functions are global, so every new function declaration is included in the Function table. The Function table stores the function's name as well as a vector consisting of its return type and argument types. If a function is overloaded, an additional vector of return type and argument types is appended to a linked list for that function's cell. In this way, multiple functions can share the same name, so long as the number and/or types of their arguments differ.
Using these three tables to track variable and function life times, the Semantic Checker checks the following features of the language:

- Variables are declared before being initialized.
- Variables are declared and initialized before being used in expressions or functions.
- Variables are not redeclared within the same scope
- Expressions evaluate to the same type as the variable they are being assigned to.
- Functions return the same type as the variable they are being assigned to.
- Functions are called with the appropriate number and types of arguments.
- Functions return values when they are defined as such, and this value matches the return value specified.

Code Generator

One of the unique aspects of the GG translator is that code generation occurs concurrently with the semantic walk of the tree. As the semantic checker walks through the tree, each node is interpreted into Java, creating a file called MainMethod.java that defines the MainMethod class. Once this class has been created and the tree has been fully walked, a wrapper file, sharing the name of the interpreted file, is created to instantiate and call the MainMethod class that we created. Additional files may also be created should any threaded functions exist, as threaded functions need to be implemented as members of separate classes.
The actual process of generating code is rather straightforward. For constructions that Java and GG share, the nodes in the AST are translated into the appropriate Java code, adding any necessary punctuation that the Lexer removed. For GG specific constructs, such as library functions or threaded functions, a little extra code is needed. Any functions that are native to guts are simply prefaced by the "guts" object that is global to our MainMethod class so that Java can recognize these functions as part of our package. Threaded functions, however, are not written to our MainMethod file. Instead, each threaded function is implemented as a new Java class that implements the thread interface. By doing this, we can then run each of these thread class's run() method whenever the threaded function is called.

Javac Compiler

To finish the entire process the Java output of the code generator is then fed directly to the Javac compiler, which will then produce a java class that can be run on any machine thanks to the cross platform capabilities that are already built into the Java Virtual Machine. Because the GG compiler will have done all previous error checking on the input code there will be no need for the user to interact with any of the Javac compiler's error messages or special functionality. It will simply be a clean translation from the Java code that was produced to usable java classes.

Test Plan

In creating the GG compiler, testing was a crucial method of ensuring not only that new functionality worked correctly, but that older functionality was not corrupted by adding new features. In order to test the GG compiler during development, a simple form of regression testing was used, consisting of three phases. Before testing, base cases, or modules, were created such that each tiny module of code was designed to exploit a specific feature of the GG language, for example variable initialization. In Phase 1, the functionality of the Lexer/Parser was tested against these base cases. Each module had one or many syntactic errors introduced into it, creating a new module that the team flagged as an intentional error test. Additionally, each tree that the Lexer should output for each test file was determined by hand and stored as <testname>-BASECASE.gg. To automate this process, a script was written to run each of the files through the Lexer/Parser, capturing the outputted tree into a file, which was then checked against the known BASECASE output, as described in the lecture slides. The result of each of these tests was then logged so that the team could ensure that all intentional error modules failed and all correct modules passed.

In Phase II, the static semantic checker was put to work. The same functional modules were used, but now had semantic errors introduced into the code. Luckily, far fewer semantic checks needed to be made for each file, as most truly egregious coding errors will be picked up by the syntactic checker. Once these error modules were established, as in Phase I, the script was run again, only this time, instead of "*diff*ing" the resulting file with the syntactic tree, we simply "*diff*ed" the resulting file with a blank file. This effectively checks to see if any errors were encountered as a blank file implies that no errors were tagged during runtime. If any of the intentionally erroneous modules pass or any correct module fail, we check the log to see what errors were generated in the resulting file to assess which semantic checks needed revision.

In the final phase, Phase III, the resulting Java code from our tree walker is checked for correctness. Unfortunately, by the time we had reached this stage, our programs had become so complex that base cases were difficult to write by hand. In addition, *diff* was all too comfortable flagging errors simply because white space wasn't consistent. Therefore, instead of automating this process, the Java code that was output was checked by hand and the actual functionality of the program was thoroughly tested. One may notice that, while regression testing was easily implemented, our testing plan did not make heavy use of integration. In the early stages of GG's development, integration was targeted as a testing procedure and the first few tests implemented integration. Small modules were written in GG, the trees were planned out, and the Java code was hand written as the files were still small. However, as most projects are prone to do, various parts of the pipeline lagged behind others, and our team found ourselves with a fully functioning Lexer/Parser, and an in progress semantic checker and code generator. At that point, we opted instead for a suite that fully tested the Lexer/Parser, even as new features were added, but tested the code generation and semantic checker fully but separately.

Lessons Learned

Jake:

- Programming languages are huge. I mean, GUI's, auto-conversion, and networks in one language? Seriously...

- Don't get too tied in to your roles. Sure, division of labor is important, but don't think that your tester should just sit around and twiddle his thumbs if there's nothing to test.

- Fewer talking meetings and more doing meetings. Granted, our language went through many changes that required reworking, but new things crop up in development much more than in planning.

- When everything else shits the bed, you can still ring the bell.

Elizabeth:

- Projects should start small, add to them when you know there's time to do so.

- There's a big learning curve with ANTLR (not enough information out there about it).

- Front end can't be done alone.

Kierstan:

- Our schedule of Liz and I doing ANTLR and our due dates for implementation was a good thing because we had a plan, but we now realize that ANTLR has a huge learning curve and it helped to have more minds trying to get over the hump. Once we understood, could write the Lexer, Parser and build the tree quickly.

- Checking the semantics of our language was the most difficult.

- Our method of extreme programming was really great!

- Our team was very close and communicated well, which helped keep a project on such a large scale in order.

Jonah:

- Even a seemingly small and compact language requires a great deal of thought and design.

- Adobe Acrobat allows you to add pages just by dragging another PDF onto the one you currently have open, which is very convenient for this particular project since our final PDF will be nearly 150 pages and will contain more than a dozen different sections come from many different sources.

- Prof. Edwards might insult you or your group during your presentation if you are not careful with what you say.

Appendix A: Code Listing

```
/*GG Programming Language
 *
 *Authors:
 * Kierstan Bell
 * Elizabeth Mutter
 * Jacob Porway
 *
 * 05/13/03
 *
 *
 */

//-------------------------------------------------
//
//GGParser.g: parses the lexical tokens, checking for
//            syntax and outputs an AST
//
//-------------------------------------------------



class GGParser extends Parser;
options {
    buildAST = true;
    exportVocab=GG;                 // Call its vocabulary "GG"
    defaultErrorHandler = true;     // Don't generate parser error handlers
    k=3;
}


tokens{

TYPENAME;VARDECL;VARLIST;FUNCDECL;ARG;ARGS;RETTYPE;BODY;IF;WHILE;ASSIGN;RETURN
;FUNCCALL;SIGNED;
}


// GG has one main function per file, so this is our starting rule.
startRule
    : (decl)*
    ;

decl
    :((typeName|"void"|"threaded") ID OPAREN)=>funcDecl
    | varDecl
    ;

funcDecl!
    :   (t:typeName|v:"void"|th:"threaded")
        i:ID
        OPAREN!
        a: args
        CPAREN!
        OCURLY!
        b:body
        CCURLY!
        {#funcDecl = #(#[FUNCDECL, "FUNC_DECL"], t, v, th, i, a, b);}
    ;


args
    : (arg (COMMA! arg)*)?


    ;

arg!
```

```
    : t:typeName i:ID
     {#arg = #(#[ARG,"ARG"],t, i);}

    ;


varDecl!
   :t:typeName v:varList SEMI!
        {#varDecl = #(#[VARDECL,"VAR_DECL"],t,v);}
   ;

varList
   :
    varInit(COMMA! varInit)*

    ;

varInit
    : ID (EQUAL^ expression)?
    ;

body!
  : (b: babyBody
    {#body = #(#[BODY,"BODY"], b);})?
    ;

babyBody
    :(varDecl | statement)+
    ;



funcCall
    : ID
        OPAREN!
            (expression (COMMA! expression)*)?
        CPAREN!
    ;

statement
    : i:ifState
        {#statement = #(#[IF, "IF"], i);}
    | w:whileState
        {#statement = #(#[WHILE,"WHILE"],w);}
    | a:assignState
        {#statement = #(#[ASSIGN,"ASSIGN"],a);}
    | r:returnState
        {#statement = #(#[RETURN,"RETURN"],r);}
    | f:funcCall SEMI!
        {#statement = #(#[FUNCCALL,"FUNC_CALL"],f);}
    ;

ifState
    : "if"! OPAREN! expression CPAREN!
        (OCURLY! body CCURLY!)
        (("else")=>"else"! (OCURLY! body CCURLY!)
        | /*empty*/
        )
    ;

whileState
    //NOTE:  Jake took out (expression)?
    : "while"! OPAREN! expression CPAREN!
        (OCURLY! body CCURLY!)
    ;
```

```
assignState
    : ID EQUAL^ expression SEMI!

    ;

returnState
    : "return"! expression SEMI!
    ;


//It's a calZone!!!!! . . . The sandwich in the middle :) AKA "The rightSide"
//How many pounds does it take to make a calZone?
calZone!
    : i:ID
        {#calZone = #i;}
    | s:STRING
        {#calZone = #s;}
    | n:NUMBER
        {#calZone = #n;}
    | c:CHAR
        {#calZone = #c;}
    | t:"true"
        {#calZone = #t;}
    | l:"false"
        {#calZone = #l;}
    | f:funcCall
        {#calZone = #(#[FUNCCALL, "FUNC_CALL"], f);}
    ;

typeName
    : typeNames
    ;

typeNames
    :     "int"
        | "string"
        | "boolean"
        | "char"
        | "file"

    ;


expression
         : relationalExpression ((AND^|OR^) relationalExpression)* ;



relationalExpression
        : addingExpression ((EE^|NE^|GT^|GTE^|LT^|LTE^) addingExpression)* ;


addingExpression
        : multiplyingExpression ((PLUS^|MINUS^) multiplyingExpression)* ;


multiplyingExpression
        : signExpression ((STAR^|BSLASH^) signExpression)* ;

signExpression
    : (PLUS^|MINUS^)? booleanNegationExpression
        ;



booleanNegationExpression
        : (NOT^)? atom ;
```

```
atom
    : calZone | OPAREN! expression CPAREN!
    ;



//--------------------------------------------------
//
//GGParser.g: parses the lexical tokens, checking for
//            syntax and outputs an AST
//
//--------------------------------------------------


class GGLexer extends Lexer;


options {
    k=2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = GG;
}


DOT                : '.';
PLUS               : '+';
MINUS              : '-';
BSLASH             : '/';
STAR               : '*';
EQUAL              : '=';
OR                 : "||";
AND                : "&&";
GT                 : '>';
LT                 : '<';
GTE                : ">=";
LTE                : "<=";
NOT                : '!';
EE                 : "==";
NE                 : "!=";
SEMI               : ';';
OPAREN             : '(';
CPAREN             : ')';
OCURLY             : '{';
CCURLY             : '}';
OSQUARE            : '[';
CSQUARE            : ']';
COMMA              : ',';
COLON              : ':';

STRING
    : '"'! ( ~('"' | '\n') | ('"'! '"'))* '"'!

    ;


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


Newline
    : ( '\n' | "\r\n" | '\r' )
```

```
        { $setType(Token.SKIP);}
        {newline(); }
    ;


ID options { testLiterals = true; }
    : ( 'a'..'z' | 'A'..'Z' | '_' ) ( 'a'..'z' | 'A'..'Z' | '_' | '0'..'9' )*
    ;


NUMBER
    : ('0'..'9')+
    ;



//----------------------------------------------------
//'.'=any character and '!'=throw away the single quotes
//----------------------------------------------------
CHAR
    : "\'"! . "\'"!

    ;


SingleCom
        :       "//"
                (~('\n'|'\r'))* ('\n'|'\r'('\n')?)
                {$setType(Token.SKIP); newline();}

        ;


MultipleCom
        :       "/*"
                (       /*      '\r' '\n' can be matched in one alternative or
by matching
                                '\r' in one iteration and '\n' in another.  I
am trying to
                                handle any flavor of newline that comes in,
but the language
                                that allows both "\r\n" and "\r" and "\n" to
all be valid
                                newline is ambiguous.  Consequently, the
resulting grammar
                                must be ambiguous.  I'm shutting this warning
off.
                         */
                        options {
                                generateAmbigWarnings=false;
                        }
                :
                        { LA(2)!='/' }? '*'
                |       '\r' '\n'               {newline();}
                |       '\r'                    {newline();}
                |       '\n'                    {newline();}
                |       ~('*'|'\n'|'\r')
                )*
                "*/"
                {$setType(Token.SKIP);}

        ;
```

```
/*GG Programming Language
 *
 *Authors:
 * Kierstan Bell
 * Elizabeth Mutter
 * Jacob Porway
 *
 * 05/13/03
 *
 *
 */




header{
    import java.util.*;
    import java.io.*;
}

class GGWalkerSemantic extends TreeParser;

options {
    importVocab = GG;
}

tokens
{  EQUALS;
}

{
    /**Hashtables**/
    // maintain variable/function declarations
    Hashtable global = new java.util.Hashtable();
    Hashtable local = new java.util.Hashtable();
    Hashtable keywords = new java.util.Hashtable();
    Hashtable functions = new java.util.Hashtable();
    FunctionHash fh = new FunctionHash();
    Hashtable gghash = new Hashtable();
    FunctionHash gh = new FunctionHash();

    //Stacks: maintain scope
    Stack varStack = new Stack();

    Vector ggv = new Vector();
    int comArgs = 0;


    /**flags**/
    boolean codeGeneration = true;
    boolean mainFound = false;
    boolean returnCheck = false;
    boolean misMatch = false;
    boolean iminfunccall = false;

    /**files**/
    String filename = "Test";
    IOStuff out = new IOStuff();
    //out.newWriter(filename);



    //if errors are found, do not make tree
    //Jonah - command-line args to hashtable?
    //remember to include file rule
```

```
    //if in gghash print guts.func(); and if main, public static void main . .
.
}


startRule
    :
//-------------------------------------------------------------
        //put pre-defined functions into keyword table
        {

            out.newWriter(filename + "MainFile.java");
            keywords.put("fgetline", "fgetline")       ;
            keywords.put("fprint", "fprint");
            keywords.put("getint", "getint");
            keywords.put("getline", "getline");
            keywords.put("getlocalhost", "getlocalhost");
            keywords.put("getTime", "getTime");
            keywords.put("print", "print");
            keywords.put("recv", "recv");
            keywords.put("send", "send");
            keywords.put("socketclose", "socketclose");
            keywords.put("socketcreate", "socketcreate");
            keywords.put("stoi", "stoi");

            //put all keywords into keyword hashtable-make sure agree with
Jonah
            keywords.put("append", "append");
            keywords.put("boolean", "boolean");
            keywords.put("cd", "cd");
            keywords.put("char", "char");
            keywords.put("else", "else");
            keywords.put("false", "false");
            keywords.put("file", "file");
            keywords.put("for", "for");
            keywords.put("if", "if");
            keywords.put("int", "int");
            keywords.put("ls", "ls");
            keywords.put("mkdir", "mkdir");
            keywords.put("overwrite", "overwrite");
            keywords.put("put", "put");
            keywords.put("return", "return");
            keywords.put("rm", "rm");
            keywords.put("rmdir", "rmdir");
            keywords.put("string", "string");
            keywords.put("threaded", "threaded");
            keywords.put("true", "true");
            keywords.put("void", "void");
            keywords.put("while", "while");

            //java's keywords
            /*
            keywords.put("abstract", "abstract");
            keywords.put("double", "double");
            keywords.put("strictfp", "strictfp");
            keywords.put("interface", "interface");
            keywords.put("super", "super");
            keywords.put("break", "break");
            keywords.put("extends", "extends");
            keywords.put("long", "long");
            keywords.put("switch", "switch");
            keywords.put("byte", "byte");
            keywords.put("final", "final");
            keywords.put("native", "native");
            keywords.put("synchronized", "synchronized");
            keywords.put("case", "case");
            keywords.put("finally", "finally");
```

```
keywords.put("new", "new");
keywords.put("this", "this");
keywords.put("catch","catch");
keywords.put("for", "for");
keywords.put("private", "private");
keywords.put("froze", "froze");
keywords.put("float", "float");
keywords.put("package", "package");
keywords.put("throw", "throw");
keywords.put("throws", "throws");
keywords.put("class", "class");
keywords.put("goto", "goto");
keywords.put("protected", "protected");
keywords.put("transient", "transient");
keywords.put("const", "const");
keywords.put("public", "public");
keywords.put("try", "try");
keywords.put("continue", "continue");
keywords.put("implements", "implements");
keywords.put("default", "default");
keywords.put("import", "import");
keywords.put("short", "short");
keywords.put("volatile", "volatile");
keywords.put("do", "do");
keywords.put("instanceof", "instanceof");
keywords.put("static", "static");
*/


//add Jonah's functions to the gghash table
ggv.add("string");
ggv.add("file");
gghash = gh.putFunction(gghash, "fgetline", ggv);


ggv.clear();
ggv.add("void");
ggv.add("file");
ggv.add("string");
ggv.add("string");
gghash = gh.putFunction(gghash, "fprint", ggv);

ggv.clear();
ggv.add("int");
gghash = gh.putFunction(gghash, "getint", ggv);

ggv.clear();
ggv.add("string");
gghash = gh.putFunction(gghash, "getline", ggv);

ggv.clear();
ggv.add("string");
gghash = gh.putFunction(gghash, "getlocalhost", ggv);

ggv.clear();
ggv.add("int");
gghash = gh.putFunction(gghash, "getTime", ggv);

ggv.clear();
ggv.add("void");
ggv.add("string");
gghash = gh.putFunction(gghash, "print", ggv);

ggv.clear();
ggv.add("string");
gghash = gh.putFunction(gghash, "recv", ggv);
```

```
            ggv.clear();
            ggv.add("string");
            ggv.add("int");
            gghash = gh.putFunction(gghash, "recv", ggv);

            ggv.clear();
            ggv.add("void");
            ggv.add("file");
            gghash = gh.putFunction(gghash, "send", ggv);

            ggv.clear();
            ggv.add("void");
            ggv.add("file");
            ggv.add("int");
            gghash = gh.putFunction(gghash, "send", ggv);

            ggv.clear();
            ggv.add("void");
            ggv.add("string");
            gghash = gh.putFunction(gghash, "send", ggv);

            ggv.clear();
            ggv.add("void");
            ggv.add("string");
            ggv.add("int");
            gghash = gh.putFunction(gghash, "send", ggv);

            ggv.clear();
            ggv.add("void");
            gghash = gh.putFunction(gghash, "socketclose", ggv);

            ggv.clear();
            ggv.add("void");
            ggv.add("int");
            gghash = gh.putFunction(gghash, "socketclose", ggv);

            ggv.clear();
            ggv.add("int");
            gghash = gh.putFunction(gghash, "socketcreate", ggv);

            ggv.clear();
            ggv.add("int");
            ggv.add("int");
            gghash = gh.putFunction(gghash, "socketcreate", ggv);

            ggv.clear();
            ggv.add("int");
            ggv.add("string");
            ggv.add("int");
            gghash = gh.putFunction(gghash, "socketcreate", ggv);

            ggv.clear();
            ggv.add("int");
            ggv.add("string");
            gghash = gh.putFunction(gghash, "stoi", ggv);

//------------------------------------------------------------------


            //initialize our class and guts object
            if(codeGeneration)
            {
                //open your files
                out.Write("import guts.*;\n");
                out.Write("public class " + filename + "MainFile { \n");
                out.Write("GG_guts guts = new GG_guts(); \n");
            }
```

```
        }

        (varDecl[global]|funcDecl)*
        {
            if(mainFound == false)
            System.err.println("\n***You have not declared a main
function***");


             if(codeGeneration)
               {
                out.Write("}"); //Close our MainMethod class
                out.closeWriter();
                out.newWriter(filename + ".java");
                 out.Write("public class " + filename + "{\n");
                 out.Write("public static void main(String args[]){\n");
                 if(comArgs == 0)
                   out.Write(filename + "MainFile m = new " + filename +
"MainFile();\n");
                  else{
                    out.Write(filename + "MainFile m = new " + filename +
"MainFile(args[0]");
                    for (int i = 1; i < comArgs; i++){
                      out.Write(", args[" + i + "]");
                    }
                    out.Write(");\n");
                    out.Write("}\n");
                    out.Write("}");
                  }
                   out.closeWriter();
                }
            }


    ;


//NOTE:  I fixed varDecl so that it does semantic checking.  Also, code
//generation should work for varDecls (mostly)
varDecl[Hashtable varHash]
{String name;Vector v = new Vector(); boolean init = false; int count = 0;}
    : #(root:VARDECL
            {AST type = root.getFirstChild();


                AST rightChildren = type.getNextSibling();

                //BCG
                if(codeGeneration)
                {
                    if((type.getText()).compareTo("file")==0)
                    out.Write("GG_File ");
                    else
                    out.Write(type.getText() + " ");
                }
                //ECG

                while(rightChildren != null){

                    //BCG
                    if(count > 0)
                    {  if(codeGeneration)
                       out.Write(", "); }

                    if (((rightChildren.getText()).compareTo("="))==0){
                        name = (rightChildren.getFirstChild()).getText();
                        init = true;
```

```
                        //BCG
                        if(codeGeneration)
                        {
                            if((type.getText()).compareTo("file")==0)
                            out.Write("GG_File ");
                            out.Write(name + " = ");
                            if((type.getText()).compareTo("file")==0)
                            out.Write(" new GG_File(");
                        }
                        //ECG

                        //Since we're assigning it, let's fall through to
                        //expr to both generate the code for the expression
and
                        //make sure the types match
                        expr((rightChildren.getFirstChild()).getNextSibling(),
type.getText());
                    }
                    else{
                        name = rightChildren.getText();

                        //BCG
                        if(codeGeneration)
                        out.Write(name);
                        //ECG
                    }

                    if(keywords.containsKey(name))
                    System.err.println("\nvariable [" + name +"]: beat you to
it... we've claimed that name");

                    else if(!varHash.containsKey(name)){
                        v.add(type.getText());
                        Boolean tb = new Boolean(init);
                        v.add(tb);
                        varHash.put(name, v);


                    }

                    else
                    System.err.println("\nI'm sorry...variable" + name + " has
already been declared");
                    count++;
                    rightChildren = rightChildren.getNextSibling();
                }

                //BCG
                if(codeGeneration)
                out.Write(";");
                //ECG

            }
        )
    ;

funcDecl
{Vector v = new Vector(); }
    :#(func:FUNCDECL
            {AST typeNode = func.getFirstChild();
                String type = new String(typeNode.getText());
                local.put("return", new String(type));
                v.add(type);
                AST funcNode = typeNode.getNextSibling();
                String key = new String(funcNode.getText());
                if(key.equals("main")){
```

```
            mainFound = true;
        }

        /***********************************************/
        if(codeGeneration){
                if(key.equals("main"))
                {
                   out.Write("public " + filename + "MainFile(");
                }

                else if(type.equals("threaded"))
                {
                out.closeWriter();
                out.newWriter(key + "ThreadedFunc.java");
                out.Write("import guts.*;\n");
                out.Write("public class " + key + "ThreadFunc{\n");
                out.Write("public " + key + "ThreadFunc(");
                out.closeWriter();
                out.newWriter(filename + "MainFile.java");
                }
                else
                {
                   if(type.equals("string"))
                   out.Write("String " + key + "(");
                   else if(type.equals("file"))
                   out.Write("GG_File " + key + "(");
                   else
                   out.Write(type + " " + key + "(");
                }
        }
        /***********************************************/

        if(keywords.containsKey(key)){
            System.err.println("\n[" + key + "]: is a reserved word
and cannot be used as a function name");
        }


        AST funcStuff = funcNode.getNextSibling();


        /*--create a vector to store the arguments of the
funcDecl---*/
        if(funcStuff != null && (funcStuff.getText()).equals("ARG")){
            String argType = (funcStuff.getFirstChild()).getText();
            String argName =
((funcStuff.getFirstChild()).getNextSibling()).getText();
            Vector varv = new Vector();
            varv.add(argType);
            varv.add("false");
            local.put(argName, varv);
            v.add(argType);

                 /***********************************************/
                if(codeGeneration){
                   if(type.equals("threaded"))
                   {
                out.closeWriter();
                out.newWriter(key + "ThreadedFunc.java");
                if(argType.equals("string"))
                out.Write("String " + argName);
                else if(argType.equals("file"))
                out.Write("GG_File " + argName);
                else
                out.Write(argType + " " + argName);
                out.closeWriter();
                out.newWriter(filename + "MainFile.java");
```

```
                        }
                        else{
                        if(argType.equals("string"))
                        out.Write("String " + argName);
                        else if(argType.equals("file"))
                        out.Write("GG_File " + argName);
                        else
                        out.Write(argType + " " + argName);
                        }
                    }
                     /************************************************/


                    funcStuff = funcStuff.getNextSibling();

                    while(funcStuff != null &&
(((funcStuff.getText()).compareTo("ARG"))==0)){
                    argType = (funcStuff.getFirstChild()).getText();
                    v.add(argType);
                    argName =
((funcStuff.getFirstChild()).getNextSibling()).getText();
                    local.put(argName, argType);
                    funcStuff = funcStuff.getNextSibling();

                     /************************************************/
                    if(codeGeneration){
                            if(type.equals("threaded"))
                        {
                            out.closeWriter();
                            out.newWriter(key + "ThreadedFunc.java");
                            if(argType.equals("string"))
                            out.Write("String " + argName);
                            else if(argType.equals("file"))
                            out.Write("GG_File " + argName);
                            else
                            out.Write(argType + " " + argName);
                            out.closeWriter();
                            out.newWriter(filename + "MainFile.java");
                        }
                        else{
                        if(argType.equals("string"))
                        out.Write("String " + argName);
                        else if(argType.equals("file"))
                        out.Write("GG_File " + argName);
                        else
                        out.Write(argType + " " + argName);
                        }

                    }
                     /************************************************/

                }
            }


/*-------------------------------------------------------------*/
            //save the number of command line arguments
            if(key.equals("main"))
                    comArgs = v.size() - 1;


            //When allowing overloading, make sure that not declared with
same number and types of args

            if(functions.containsKey(key)){
                LinkedList declareFunc = new LinkedList();
```

```
                    declareFunc = (LinkedList)functions.get(key);
                    Iterator iter = declareFunc.iterator();
                    boolean same = true;
                    int i = 1;
                    while (iter.hasNext()){
                        Vector checkv = (Vector)iter.next();
                        for (i=1; i < (checkv.size()-1); i++){
                            if(v.elementAt(i) != null){

if(!(((String)v.elementAt(i)).equals((String)(checkv.elementAt(i))))){
                                    same = false;
                                }
                            }
                        }
                        if( ((same == true) && (v.size() > checkv.size())) ||
(v.size() < checkv.size()) )
                        same = false;
                    }

                    if((same == true))
                    System.err.println("function [" + key + "]:I'm sorry . . .
no, you cannot redeclare this function!");
                }


                functions = fh.putFunction(functions, key,v);



                /************************************************/
                if(codeGeneration)
                {
                if(type.equals("threaded")){
                        out.closeWriter();
                        out.newWriter(key + "ThreadedFunc.java");
                        out.Write(") {\n");
                        out.Write("GG_guts guts = new GG_guts\n");
                        out.closeWriter();
                        out.newWriter(filename + "MainFile.java");
                }
                }
                else{
                  out.Write(") { \n");

                }
                /************************************************/


                /**BODY**/
                if (funcStuff != null){
                if(type.equals("threaded")){
                out.closeWriter();
                out.newWriter(key + "ThreadedFunc.java");
}

                   body(funcStuff);
                out.closeWriter();
                out.newWriter(filename + "MainFile.java");

                }

                /************************************************/
                if(codeGeneration)
                {
                if(type.equals("threaded")){
                out.closeWriter();
                out.newWriter(key +"ThreadedFunc.java");
```

```
                out.Write("} \n");
                out.Write("} \n");
                out.closeWriter();
                out.newWriter(filename + "MainFile.java");
                out.Write(key + "ThreadFunc " + key + "ThreadObj = new " + key
+ "ThreadedFunc (this);\n");
                }
                else{
                  out.Write("} \n");
                }
                }
                /***********************************************/


                if(!((String)(local.get("return"))).equals("void") &&
!((String)(local.get("return"))).equals("threaded") && returnCheck != true)
                System.err.println("function [" + key + "]: Expecting a return
statement");
                if(((String)(local.get("return"))).equals("void") &&
returnCheck == true)
                System.err.println("function [" + key + "]: This function
returns void, not expecting a return statement");
                if(((String)(local.get("return"))).equals("threaded") &&
returnCheck == true)
                System.err.println("function [" + key + "]: Threaded functions
can not return anything");
                local.clear();
                returnCheck = false;


            }
        )
    ;



expr[String mytype]
{String ourType = "";}

    :ID {
            //BCG
            if(codeGeneration)
                out.Write(#ID.getText());
            //ECG

            if(!global.containsKey(#ID.getText()) &&
!local.containsKey(#ID.getText()) || mytype.compareTo("")==0)
             {
                if(iminfunccall == true)
                misMatch = true;
                else
                System.err.println("\nvariable [" + #ID.getText() + "]: The
thing is...I don't know what " + #ID.getText() + " is, you need to fill me on
that first");

            }

            else {

                if (local.containsKey(#ID.getText())){
                  Vector v = (Vector)local.get(#ID.getText());
                    if(v.elementAt(1).equals(new Boolean(true)))
                    {
                        ourType = (String)v.elementAt(0);
                        if (!(ourType.equals(mytype)))
                         {
```

```
                        if(iminfunccall == true)
                        misMatch = true;
                        else
                        System.err.println("variable [" + #ID.getText() +
"]:is " + ourType +" but should be " + mytype);
                    }
                }
                else
                {
                    if(iminfunccall == true)
                    misMatch = true;
                    else
                    System.err.println("variable [" + #ID.getText() + "]:
you haven't initialized this guy locally");
                }
            }
            else{
                Vector v = (Vector)global.get(#ID.getText());
                if( v.elementAt(1).equals(new Boolean(true))){
                    ourType = (String)v.elementAt(0);
                    if(!(ourType.equals(mytype)))
                    {
                        if(iminfunccall == true)
                        misMatch = true;
                        else
                        System.err.println("variable [" + #ID.getText() +
"]:is " + ourType +" but should be " + mytype);
                    }
                }
                else
                {
                    if(iminfunccall == true)
                    misMatch = true;
                    else
                    System.err.println("variable [" + #ID.getText() + "]:
you haven't initialized this guy globally");
                }
            }


        }
    }


| NUMBER
    {
        if(!(mytype.equals("int")))
        {
            if(iminfunccall == true)
            misMatch = true;
            else
            System.err.println("[]: expected " + mytype + " got " +
#NUMBER.getText());
        }
        //BCG
        if(codeGeneration)
            out.Write(#NUMBER.getText());
        //ECG
    }


| STRING{
        if(!(mytype.equals("string")))
        {
            if(iminfunccall == true)
            misMatch = true;
            else
```

```
                System.err.println("[]: expected " + mytype + " got " +
#STRING.getText());
            }
            //BCG
            if(codeGeneration)
        out.Write(#STRING.getText());
            //ECG
        }
    | CHAR {
            if(!(mytype.equals("char")))
             {
                if(iminfunccall == true)
                misMatch = true;
                else
                System.err.println("[]: expected " + mytype + " got " +
#CHAR.getText());
            }
            //BCG
            if(codeGeneration)
        out.Write(#CHAR.getText());
            //ECG
        }


    | FILE {
            if(!(mytype.equals("file")))
            {
                if(iminfunccall == true)
                misMatch = true;
                else
                System.err.println("[]: expected " + mytype + " got " +
#FILE.getText());
            }
            //BCG
            if(codeGeneration)
        out.Write(#FILE.getText());
            //ECG
        }

    | "true"{
            if(!(mytype.equals("boolean")))
            {
                if(iminfunccall == true)
                misMatch = true;
                else
                System.err.println("[True]: expected " + mytype + " got
true");
            }
            //BCG
            if(codeGeneration)
        out.Write("true");
            //ECG
        }
    | "false"{
            if(!(mytype.equals("boolean")))
            {
                if(iminfunccall == true)
                misMatch = true;
                else
                System.err.println("[False]: expected " + mytype + " got
false");
            }
            //BCG
            if(codeGeneration)
        out.Write("false");
            //ECG
```

```
        }
    | #(root0:FUNCCALL

//**************************************************************************
***********

            {//remember to check for the empty string and know that is can't
be assigned


                iminfunccall = true;
                AST leftChild = root0.getFirstChild();
                String key = new String(leftChild.getText());
                Vector v = new Vector();
                misMatch = false;
                String rettype = "";

                if(functions.containsKey(key) || gghash.containsKey(key))
                {
                    LinkedList declareFunc = new LinkedList();

                    if(functions.containsKey(key))
                    declareFunc = (LinkedList)functions.get(key);
                    else
                    declareFunc = (LinkedList)gghash.get(key);

                    Iterator iter = declareFunc.iterator();
                    int i = 1;
                    boolean foundMatch = false;
                    AST nextArg =leftChild.getNextSibling();

                    while(nextArg != null)
                    {
                        v.add(nextArg);
                        nextArg = nextArg.getNextSibling();
                    }



                    //loop through the functions of the linked list til you
find a match
                    while (iter.hasNext() && foundMatch == false)
                    {

                        //checkv:current function in the linked list you're
looking at
                        Vector checkv = (Vector)iter.next();

                        //thistype:return type of the current function
                        String thistype = (String)checkv.elementAt(0);

                        //if the return types don't match, this function
cannot possibly be a match
                        //so first check that they do, and then check the args
                        if( mytype.equals(thistype) || mytype.equals("") )
                        {

                            //loop through all the args of the current
function
                            for (i=1; i < (checkv.size()-1); i++)
                            {

                                //check that the funccall has a new arg to
check
                                if(v.elementAt(i-1) != null)
                                {
                                    //pass the funccall arg and the type of
```

```
the current arg  to expr to check types
                                    expr((AST)v.elementAt(i-1),
(String)checkv.elementAt(i));
                                    //keep track of how many funccall args you
get through

                                }//if



                            }//end for: out of one function of the linked list

                            //if: no mismatches, #funccall args = #current
func's args, & fuccall doesnt have more args


                           if( (misMatch == false) && (v.size() ==
checkv.size()-1)){
                                foundMatch = true;
                                rettype = (String)checkv.elementAt(0);

                            }
                        }//if


                    }//while


                    if(foundMatch == false)
                    {
            System.out.println("function call [" + key + "()]: i don't think
this exists...you should probably check that you're passing the right
things");
                    }

                }//big if

                else
                    System.out.println("function [" + key + "]: not
declared");

                  /************************************************/
                    if(codeGeneration)
                    {
                        if(gghash.containsKey(key))
                        {
                           out.Write("guts.");
                        }

                        if(rettype.equals("threaded"))
                        {
                           out.Write(key + "ThreadObj.");
                        }

                        out.Write(key + "(");
                        if(!v.isEmpty())
                        {
                            out.Write((String)v.elementAt(0));
                            for(int i = 1; i < v.size(); i++)
                            {
                               out.Write(", " + (String)v.elementAt(i));

                            }
                        }
                        out.Write(");\n");

                    }
```

```
                /***********************************************/

                //clear flags
                misMatch = false;
                iminfunccall = false;
            }
        )

//*************************************************************************
*************
    | #(root1:EQUAL
            {AST leftChild = root1.getFirstChild();
                if(local.containsKey(leftChild.getText())){
                    Vector v = (Vector)local.get(leftChild.getText());
                    v.setElementAt(new Boolean(true), 1);
                }
                else if(global.containsKey(leftChild.getText())){
                    Vector v = (Vector)global.get(leftChild.getText());
                    v.setElementAt(new Boolean(true), 1);
                }


                expr(leftChild, mytype);

                //BCG
                if(codeGeneration)
                {
                    out.Write(" = ");
                    if(mytype.compareTo("file")==0)
                    out.Write("new GG_File(");
                }
                //ECG


                expr(leftChild.getNextSibling(), mytype);


            }

        )
    | #(root2:PLUS
            {AST leftChild = root2.getFirstChild();


                if(mytype.compareTo("int") != 0)
                System.err.println("\n[?]  This expression returns an int, but
you're trying to assign it to a " + mytype);

                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" + ");
                //ECG

                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG
```

```
            }

        )
    | #(root3:STAR
            {
                AST leftChild = root3.getFirstChild();

                 if(mytype.compareTo("int") != 0)
                System.err.println("\n[?]  This expression returns an int, but
you're trying to assign it to a " + mytype);

                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" * ");
                //ECG

                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG

            }
        )
    | #(root4:BSLASH
            {
                AST leftChild = root4.getFirstChild();

                 if(mytype.compareTo("int") != 0)
                System.err.println("\n[?]  This expression returns an int, but
you're trying to assign it to a " + mytype);

                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" / ");
                //ECG

                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG

            }
        )
    | #(root5:OR
            {

                AST leftChild = root5.getFirstChild();

                if(!mytype.equals("boolean"))
```

```
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);

                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "boolean");

                //BCG
                if(codeGeneration)
                out.Write(" + ");
                //ECG

                expr(leftChild.getNextSibling(), "boolean");

            }
        )
    | #(root6:AND
            {
                AST leftChild = root6.getFirstChild();


                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);

//BCG
                if(codeGeneration)
                out.Write("(");
                //ECG
                expr(leftChild, "boolean");

                //BCG
                if(codeGeneration)
                out.Write(" + ");
                //ECG

                expr(leftChild.getNextSibling(), "boolean");
            }
        )
    | #(root7:GT
            {
                AST leftChild = root7.getFirstChild();

                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);

                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" > ");
                //ECG


                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
```

```
                out.Write(")");
                //ECG
            }
        )
    | #(root8:GTE
            {
                AST leftChild = root8.getFirstChild();

                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);

                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" >= ");
                //ECG

                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG
            }
        )
    | #(root9:LT
            {
                AST leftChild = root9.getFirstChild();

                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);
                //BCG
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" < ");
                //ECG

                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG
            }
        )
    | #(root10:LTE
            {
                AST leftChild = root10.getFirstChild();

                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);
                //BCG
```

```
                if(codeGeneration)
                out.Write("(");
                //ECG

                expr(leftChild, "int");

                //BCG
                if(codeGeneration)
                out.Write(" <= ");
                //ECG

                expr(leftChild.getNextSibling(), "int");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG
            }
        )
    | #(root11:NOT
            {AST leftChild = root11.getFirstChild();

                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);
                //BCG
                if(codeGeneration)
                out.Write(" !(");
                //ECG

                expr(leftChild, "boolean");

                //BCG
                if(codeGeneration)
                out.Write(")");
                //ECG
            }
        )
    | #(root12:NE
            {

                //Let's just bypass this node altogether.  Instead of doing a
!= b, let's just always do !(a == b).
                //This will make code generation much, much easier for files
and strings.  P.S. - We could probably
                //just change this in the parser and get rid of this rule once
and for all..

                AST equalsNode = #(#[EE, "EE"]);
                equalsNode.addChild(root12.getFirstChild());

equalsNode.addChild((root12.getFirstChild()).getNextSibling());
                AST newNode = #(#[NOT, "NOT"]);
                newNode.addChild(equalsNode);
                expr(newNode, "boolean");


            }
        )
    | #(root13:EE
            {
                AST leftChild = root13.getFirstChild();

                if(!mytype.equals("boolean"))
                System.err.println("\n[?]  This expression returns a boolean,
but you're trying to assign it to a " + mytype);
```

```
                int typeNum = leftChild.getType();
                String typeName = leftChild.getText();

                //This is a wee-bit messy, but I wasn't sure how else to do
this.  We need to check what kind of
                //node we have on the left, but just checking the text might
throw back "a" as a variable.  What
                //you'd like instead is to know that it's of type ID.
Therefore, I just figured out the type numbers
                //for each token and used those...

                //Okay, so we've checked if the return type is boolean, but
now we have to reset mytype to match
                //whatever the type of the left side is

                //If we've got an operator that returns ints

//if((typeMatch.compareTo("+")==0)||(typeMatch.compareTo("-")==0)||(typeMatch.
compareTo("*")==0)||(typeMatch.compareTo("/")==0))
                if(typeNum >= 49 && typeNum <= 52)
                    mytype = "int";

                //If we've got an operator that returns booleans
                //else
if((typeMatch.compareTo("<")==0)||(typeMatch.compareTo(">")==0)||(typeMatch.co
mpareTo("<=")==0)||(typeMatch.compareTo(">=")==0)||(typeMatch.compareTo("&&")=
=0)||(typeMatch.compareTo("||")==0)||(typeMatch.compareTo("!")==0)||(typeMatch
.compareTo("==")==0)||(typeMatch.compareTo("!=")==0))

                else if((typeNum >= 41 && typeNum <= 48) || typeNum == 53)
                mytype = "boolean";

                //If we've got an ID
                else if(typeNum == 19)
                    {
                    if (local.containsKey(typeName)){
                        Vector v = (Vector)local.get(typeName);
                        mytype = (String)v.elementAt(0);
                    }
                    else if(global.containsKey(typeName)){
                        Vector v = (Vector)global.get(typeName);
                        mytype = (String)v.elementAt(0);
                    }
                    else
                        mytype = "undeclared variable";
                }

                //else if(typeNum == 16)
                //{




                //If we've got a literal
                else if(typeNum == 32)  mytype = "int";
                else if(typeNum == 33)  mytype = "string";
                else if(typeNum == 31)    mytype = "char";
                else if(typeName.compareTo("true")==0 ||
typeName.compareTo("false")==0) mytype = "boolean";


                //BCG
                if(codeGeneration)
                {

                    if(mytype.compareTo("file")==0 ||
```

```
mytype.compareTo("string")==0)
                    out.Write(".equals( ");
                    else
                    out.Write(" == ");
                }
                //ECG

                        expr(leftChild.getNextSibling(), mytype);

                //BCG
                if(codeGeneration)
                {
                    if(mytype.compareTo("file")==0 ||
mytype.compareTo("string")==0)
                    out.Write(")");
                }
                //ECG

            }
        )
    | #(root14:MINUS
            {if (((root14.getFirstChild()).getNextSibling()) != null){
                    AST leftChild = root14.getFirstChild();

                     if(mytype.compareTo("int") != 0)
                System.err.println("\n[?]  This expression returns an int, but
you're trying to assign it to a " + mytype);

                    //BCG
                    if(codeGeneration)
                        out.Write("(");
                    //ECG

                    expr(leftChild, "int");

                    //BCG
                    if(codeGeneration)
                    out.Write(" - ");
                    //ECG

                    expr(leftChild.getNextSibling(), "int");

                    //BCG
                    if(codeGeneration)
                    out.Write(")");
                    //ECG
                }

                else{

                    //BCG
                    if(codeGeneration)
                    out.Write(" -(");
                    //ECG
                    AST leftChild = root14.getFirstChild();
                    expr(leftChild, "int");

                    //BCG
                    if(codeGeneration)
                    out.Write(")");
                    //ECG
                }
            }
        )
    ;
```

```
statement
{String ourType = "";}
    : #(ifRoot:IF
            {
                //BCG
                if(codeGeneration)
                out.Write("if( ");
                //ECG

                AST leftChild = ifRoot.getFirstChild();
                expr(leftChild, "boolean");

                varStack.push(local.clone());

                //BCG
                if(codeGeneration)
                out.Write(")\n{");
                //ECG

                body(leftChild.getNextSibling());

                //BCG
                if(codeGeneration)
                out.Write("}");
                //ECG

                local = (Hashtable)varStack.pop();


                AST rightChild = leftChild.getNextSibling();

                 //Check for an else block
                if(rightChild.getNextSibling() != null)
                {

                    //BCG
                    if(codeGeneration)
                    out.Write("else {");
                    //ECG

                    varStack.push(local.clone());
                    body(rightChild.getNextSibling());
                    local = (Hashtable)varStack.pop();

                    //BCG
                    if(codeGeneration)
                    out.Write("}");
                    //ECG

                }
            }
        )
    |#(whileRoot:WHILE
            {
                //BCG
                if(codeGeneration)
                out.Write("while( ");
                //ECG


                AST leftChild = whileRoot.getFirstChild();
                expr(leftChild, "boolean");

                varStack.push(local.clone());

                //BCG
                if(codeGeneration)
```

```
                out.Write(")\n{");
                //ECG

                body(leftChild.getNextSibling());


                //BCG
                if(codeGeneration)
                out.Write("}");
                //ECG

                local = (Hashtable)varStack.pop();


            }

        )
    |#(assignRoot:ASSIGN
            {AST firstChild = (assignRoot.getFirstChild()).getFirstChild();
                if(global.containsKey(firstChild.getText()) ||
local.containsKey(firstChild.getText()))
                {
                    if (local.containsKey(firstChild.getText())){
                        Vector v = (Vector)local.get(firstChild.getText());
                        ourType = (String)v.elementAt(0);
                    }
                    else{
                        Vector v = (Vector)global.get(firstChild.getText());
                        ourType = (String)v.elementAt(0);
                    }
                }

                expr(assignRoot.getFirstChild(), ourType);

                //BCG
                if(codeGeneration)
                out.Write(";");
                //ECG

            }
        )
    |#(returnRoot:RETURN
            {
                //BCG
                if(codeGeneration)
                out.Write("return ");
                //ECG

                returnCheck = true;
                ourType = (String)local.get("return");
                AST leftChild = returnRoot.getFirstChild();
                expr(leftChild, ourType);

                //BCG
                if(codeGeneration)
                out.Write(";");
                //ECG

            }
        )
    |#(funcRoot:FUNCCALL
            {
                expr(funcRoot, ourType);

                //BCG
                if(codeGeneration)
                out.Write(";");
```

```
                //ECG

            }
        )
    ;

body
    :#(BODY ( varDecl[local] | statement)*
        )
    ;

arg
    :#(argRoot:ARG
            {AST leftChild = argRoot.getFirstChild();
                out.Write(leftChild.getText() + " ");
                leftChild = leftChild.getNextSibling();
                out.Write(leftChild.getText());
            }
        )
    ;

//"I can't keep track of these children" ~Elizabeth
//"Wait . . . do you call body on the funcStuff" ~Elizabeth
//"oooooh . . . collections are double things in a bucket" ~Kierstan
```

```
/*GG Programming Language
 *
 *Authors:
 * Kierstan Bell
 * Elizabeth Mutter
 * Jacob Porway
 *
 * 05/13/03
 *
 *
 */


//-------------------------------------------------
//
//Main: compiles the GG lexer/parser/walker
//
//-------------------------------------------------

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;

class Main{

    public static void main(String[] args){

        if(args.length != 1)
            {
                System.err.println("USAGE: java Main <fileName>.gg" );
                System.exit(0);
            }

        try{

            BufferedReader  r = new BufferedReader(new FileReader(args[0]));
            GGLexer  lexer = new GGLexer(r);
            lexer.setFilename(args[0]);

            GGParser  parser = new GGParser(lexer);
            parser.startRule();
            CommonAST  t = (CommonAST) parser.getAST();
            System.out.println(t.toStringList());
            GGWalker  walker = new GGWalker();
            walker.startRule(parser.getAST());


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

}//class main
```

```
/*GGProgramming Language
 *
 *Authors:
 *Kierstan Bell
 *Elizabeth Mutter
 *Jacob Porway
 *
 *05/13/03
 *
 */

//-------------------------------------------------
//
//Puts Linked Lists into a hashtable for collision
//handling. This is to make function overloading
//possible.
//
//-------------------------------------------------


import java.util.*;

public class FunctionHash
{
    LinkedList bucket;

    public FunctionHash(){
        bucket = new LinkedList();
    }

    public Hashtable putFunction(Hashtable hash, String key, Vector argList)
    {
        bucket = new LinkedList();

        if(hash.containsKey(key))
            bucket = (LinkedList)hash.get(key);
        else
            bucket.clear();

        bucket.addFirst(argList);

        hash.put(key, (LinkedList)bucket);

        return hash;
    }

}
```

```
/*GGProgramming Language
 *
 *Authors:
 *Kierstan Bell
 *Elizabeth Mutter
 *Jacob Porway
 *
 *05/13/03
 *
 */

//-------------------------------------------------
//
//File writer methods for writing the generated
//java code to its associated file.
//
//-------------------------------------------------


import java.io.*;

public class IOStuff{

    public IOStuff(){

    }

    public void newWriter(String filename){
        try{
            System.out.println(filename);
            runF = new File(filename);
            runFile = new FileWriter(runF, true);
        }
        catch(IOException e){
            System.err.println("io exception: " + e);
        }
    }
    public void Write(String text) {
        try{
            System.out.println("wrote to file: " + text);
            runFile.write(text);
        }
        catch(IOException e){
            System.err.println("io exception: " + e);
        }
    }

    public void closeWriter(){
        try{
            runFile.close();
        }
        catch(IOException e){
            System.err.println("io exception: " + e);
        }
    }


    File runF;
    FileWriter runFile;

}
```

```
package guts;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */

import java.io.*;
import java.util.*;
import java.net.*;

public class GG_guts {
  Calendar theBeginning;
  GG_socket sockets[];

  /**
   * Default constructor for the GG_guts class sets up the array of GG_sockets
   */
  public GG_guts()
  {
    theBeginning = Calendar.getInstance();
    //sets the date to Feb 9, 2003 @ 21:00
    theBeginning.set(2003, 2, 9, 21, 0, 0);

    sockets = new GG_socket[65536];
    for(int i=0; i<sockets.length; i++)
    {
      sockets[i] = null;
    }
  }

  //General Functions
  /**
   * Prints a line of text to the specified file.
   *
   * @param f the file to write to.
   * @param option either "append" or "overwirte"
   * @param line the line of text to write to the file
   */
  public void fprint(GG_file f, String option, String line)
  {
    if(option.equals("append"))
    {
      f.appendContent(line);
    }
    else if(option.equals("overwrite"))
    {
      f.writeContent(line);
    }
    else
    {
      System.out.println("Should never have gotten to this point; bad call to "   +
                         "fprint()");
    }
  }

  /**
   * Reads the next line of text from a file.
   *
```

```
   * @param f the file to read from.
   * @return a string containing the text read from the file.
   */
  public String fgetline(GG_file f)
  {
    return f.getline();
  }

  /**
   * Writes a line of text to standard out.
   *
   * @param line the line of text to be written.
   */
  public void print(String line)
  {
    System.out.print(line);
    System.out.flush();
  }

  /**
   * Reads a line of text from standard in.
   *
   * @return a string containing the line of text.
   */
  public String getline()
  {
    String line = "";

    try
    {
      BufferedReader lineIn = new BufferedReader(new InputStreamReader(System.
          in));
      line = lineIn.readLine();
    }
    catch(IOException e)
    {
      System.err.println("Could not read from command line." );
    }

    return line;
  }

  /**
   * Reads an int from standard in.
   *
   * @return the read int.
   */
  public int getint()
  {
    int input = 0;
    String line = "";

    try
    {
      BufferedReader lineIn = new BufferedReader(new InputStreamReader(System.
          in));
      line = lineIn.readLine();
      line = (new StringTokenizer(line)).nextToken();
      input = (new Integer(line)).intValue();
    }
    catch(IOException e)
    {
      System.err.println("Could not read from command line." );
```

```
      System.err.println(e);
    }

    return input;
  }

  /**
   * Converts a string representation of an integer to an int.
   *
   * @param s string representation of the integer.
   * @return value of integer represented by the input string.
   */
  public int stoi(String s)
  {
    return (new Integer(s)).intValue();
  }

  /**
   * Converts an int to a string.
   *
   * @param i int value to be converted.
   * @return string representation of the int argument.
   */
  public String itos(int i)
  {
    return "" + i;
  }

  /**
   * Returns the number of seconds that have passed since February 9, 2003 at
   * 9:00pm EST.
   *
   * @return seconds.
   */
  public int getTime()
  {
    return (int)((System.currentTimeMillis() -
                  theBeginning.getTimeInMillis())/1000);
  }


  // Network Initialization Functions
  /**
   * Opens the next available port as a server socket.
   *
   * @return the port for the socket opened.
   */
  public int sockcreate()
  {
    GG_socket sock = new GG_socket();
    if(sock.getLocalPort() != -1)
    {
      sockets[sock.getLocalPort()] = sock;
    }

    return sock.getLocalPort();
  }
  /**
   * Atempts to open the socket for the port specified.
   *
   * @param port the port to open the socket on.
   * @return the port number or -1 on failure.
   */
```

```
public int sockcreate(int port)
{
  GG_socket sock = new GG_socket(port);
  sockets[sock.getLocalPort()] = sock;

  return sock.getLocalPort();
}

/**
 * Waits for an incoming socket connection on the specified port.
 *
 * @param port the port number for the socket to listen on.
 * @return the port number for the local connection port or -1 on failure.
 */
public int socketconnect(int port)
{
  int status = -1;
  if(sockets[port] != null)
  {
    status = sockets[port].acceptIncoming();
  }
  return status;
}

/**
 * Connects to a remote system's listening port.
 *
 * @param host hostname or IP address for the remote system.
 * @param port the port on the remote system to connect to.
 * @return the port number of the local socket connection.
 */
public int socketconnect(String host, int port)
{
  GG_socket sock = new GG_socket(host, port);
  sockets[sock.getLocalPort()] = sock;

  return sock.getLocalPort();
}

/**
 * Returns the local host IP address.
 *
 * @return local IP address as a string.
 */
public String getlocalhost()
{
  try
  {
    return (InetAddress.getLocalHost()).getHostAddress();
  }
  catch(UnknownHostException e)
  {
    System.err.println("UnknownHostException occur while getting local "   +
                       "hostname.");
    return "";
  }
}


//Socket Functions
/**
 * Sends a line of text to all connected sockets.
 *
```

```
   * @param line text to send.
   */
  public void send(String line)
  {
    for(int i=0; i < sockets.length; i++)
    {
      if(sockets[i] != null)
      {
        sockets[i].send(line);
      }
    }
  }
  /**
   * Sends a line of text to the specified socket connection.
   *
   * @param line text to send.
   * @param port port number for socket to transmit on.
   */
  public void send(String line, int port)
  {
    if(sockets[port] != null)
    {
      sockets[port].send(line);
    }
    else
    {
      System.err.println("Did not find port "  + port);
    }
  }

  /**
   * Sends the contents of a file to all connected sockets.
   *
   * @param file the file to get the contents from.
   */
  public void send(GG_file file)
  {
    for(int i=0; i < sockets.length; i++)
    {
      if(sockets[i] != null)
      {
        sockets[i].send(file.getContent());
      }
    }
  }
  /**
   * Sends the contents of a file to a specified socket connection.
   *
   * @param file the file to get the contents from.
   * @param port the port number of the socket conection to send the file
   * contents to.
   */
  public void send(GG_file file, int port)
  {
    if(sockets[port] != null)
    {
      sockets[port].send(file.getContent());
    }
  }

  /**
   * Gets a line of text from the first open socket connection.
   *
```

```
   * @return the line of text in a string.
   */
  public String recv()
  {
    String line = "";
    int i = 0;

    while(sockets[i] == null && i < sockets.length)
    {
      i++;
    }

    if(sockets[i] != null)
    {
      line = sockets[i].recv();
    }

    return line;
  }
  /**
   * Gets a line of text from a specified open socket connection.
   *
   * @param port the port number for the socket to get the line of text from.
   * @return the text in a string.
   */
  public String recv(int port)
  {
    String line = "";

    if(sockets[port] != null)
    {
      line = sockets[port].recv();
    }

    return line;
  }


  //Socket termination methods
  /**
   * Closes all open sockets.
   */
  public void sockclose()
  {
    for(int i=0; i < sockets.length; i++)
    {
      if(sockets[i] != null)
      {
        sockets[i].close();
        sockets[i] = null;
      }
    }
  }
  /**
   * Closes the specified open socket.
   *
   * @param port the port number for the socket to be closed.
   */
  public void sockclose(int port)
  {
    if(sockets[port] != null)
    {
      sockets[port].close();
```

```
      sockets[port] = null;
    }
  }
}
```

```
package guts;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author unascribed
 * @version 1.0
 */

import java.net.*;
import java.io.*;

public class GG_socket
{
  Socket me;
  ServerSocket  sSock;

  BufferedReader  in;
  BufferedWriter  out;

  /**
   * Sets up the GG_socket object with any availabe port.
   */
  public GG_socket()
  {
    try
    {
      sSock = new ServerSocket(0);
      me = null;
      in = null;
      out = null;
    }
    catch(IOException  e)
    {
      System.err.println("Error occured while opening server socket"  );
    }
  }

  /**
   * Sets up the GG_socket object with the specified port.
   *
   * @param port the port to setup the socket on.
   */
  public GG_socket(int port)
  {
    try
    {
      sSock = new ServerSocket(port);
      me = null;
      in = null;
      out = null;
    }
    catch(IOException  e)
    {
      System.err.println("Error occured while opening server socket on port "
                         + port);
    }
  }

  /**
   * Sets up the GG_socket object by making a connection to a remote host via
```

```
   * the parameters specified.
   *
   * @param host hostname of the remote host.
   * @param port port number for the remote connection.
   */
  public GG_socket(String host, int port)
  {
    try
    {
      me = new Socket(host, port);

      in = new BufferedReader(new InputStreamReader(me.getInputStream()));
      out = new BufferedWriter(new OutputStreamWriter(me.getOutputStream()));
    }
    catch(Exception e)
    {
      System.err.println(e + " occured while trying to connect with socket."  );
    }
  }

  /**
   * Waits for a remote connection to be made.
   *
   * @return the local port number for the connection.
   */
  public int acceptIncoming()
  {
    try
    {
      me = sSock.accept();

      in = new BufferedReader(new InputStreamReader(me.getInputStream()));
      out = new BufferedWriter(new OutputStreamWriter(me.getOutputStream()));

      return me.getLocalPort();
    }
    catch(IOException e)
    {
      System.err.println("IOException occured while accepting socket connection."  );
      return -1;
    }
  }

  /**
   * Closes the socket connection.
   */
  public void close()
  {
    try
    {
      me.close();
    }
    catch(IOException e)
    {
      System.err.println("IOException occured while closing socket."  );
    }
  }


  /**
   * Gets the host name of the remote machine.
   *
   * @return the remote machines host name in a string.
```

```
   */
  public String getHostName()
  {
    if(me != null)
    {
      return (me.getInetAddress()).getHostName();
    }
    else
    {
      return null;
    }
  }

  /**
   * Gets the port of the remote socket.
   *
   * @return port number of remote socket.
   */
  public int getPort()
  {
    if(me != null)
    {
      return me.getPort();
    }
    else
    {
      return -1;
    }
  }


  /**
   * Gets the local hostname.
   *
   * @return localhost.
   */
  public String getLocalHost()
  {
    if(me != null)
    {
      return (me.getLocalAddress()).getHostAddress();
    }
    else if(sSock != null)
    {
      return (sSock.getInetAddress()).getHostAddress();
    }
    else
    {
      return null;
    }
  }

  /**
   * Gets local port number.
   *
   * @return local port.
   */
  public int getLocalPort()
  {
    if(me != null)
    {
      return me.getLocalPort();
    }
```

```
    else if(sSock != null)
    {
      return sSock.getLocalPort();
    }
    else
    {
      return -1;
    }
  }


  /**
   * Sends a line of text across the socket connection.
   *
   * @param message the text to be sent.
   */
  public void send(String message)
  {
    if(out != null)
    {
      try
      {
        out.write(message);
        out.flush();
//        System.out.println("Sending: " + message + " on port: " + me.getLocalPort());
      }
      catch(IOException e)
      {
        System.err.println("IOException occured while sending the following "   +
                           "message: "  + message);

      }
    }
    else
    {
      System.err.println("Out was null!");
    }
  }

  /**
   * Gets all the text in the incoming buffer of the socket.
   *
   * @return a string with the text in it.
   */
  public String recv()
  {
    String incoming = "";

    if(in != null)
    {
      try
      {
        while(in.ready())
        {
          incoming += ("" + (char)in.read());
        }
      }
      catch(IOException e)
      {
        System.err.println("IOException occured while reading from socket."  );
      }
    }
    else
```

```
    {
      System.err.println("In was null");
    }

    return incoming;
  }
}
```

```
package guts;

/**
 * <p>Title: File</p>
 * <p>Description: This is a file wrapper class to create file objects that
 * behave in a way which is in line with the specifications of the GG
 * programming language.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: Columbia University</p>
 * @author Jonah P. Tower
 * @version 1.0
 */

import java.io.*;

public class GG_file
{
  private File me;
  private BufferedReader  in;
  private BufferedWriter  out;

  private int currentLine ;

  /**
   * This is the constructor for the GG_file class. The constructor will open
   * the file if it exists or create a new file with the name that is given if
   * it does not.
   *
   * @param pathname the name of the file either the full path name or the
   * relative path from the location in which the program is running.
   */
  public GG_file(String pathname )
  {
    me = new File(pathname );

    try
    {
      if (!me.exists())
        me.createNewFile ();
    }
    catch(IOException  e)
    {
      System.err.println("Unable to create file with the following "   +
                         "pathname: "  + me.getAbsolutePath ());
    }

    currentLine  = 1;
  }


  /**
   * Gets the next line in the file and then increments the counter so that
   * next time the method is call the following line is returned.
   *
   * @return the next line of the file
   */
  public String getline()
  {
    String line = new String();

    try
    {
      in = new BufferedReader (new FileReader (me));
```

```
    }
    catch(FileNotFoundException  e)
    {
      System.err.println("Unable to open file "  + me.getAbsolutePath() +
                         " for reading");
    }

    try
    {
      for(int i = 0; i < currentLine; i++)
      {
        line = in.readLine();
      }
      currentLine++;

      in.close();
    }
    catch(IOException  e)
    {
      System.err.println("Unable to read from "  + me.getAbsolutePath());
    }

    if(line == null)
    {
      currentLine  = 1;

      try
      {
        in = new BufferedReader(new FileReader(me));
      }
      catch(FileNotFoundException  e)
      {
        System.err.println("Unable to open file "  + me.getAbsolutePath() +
                           " for reading");
      }

      try
      {
        for(int i = 0; i < currentLine; i++)
        {
          line = in.readLine();
        }
        currentLine++;

        in.close();
      }
      catch(IOException  e)
      {
        System.err.println("Unable to read from "  + me.getAbsolutePath());
      }
    }

    return line;
  }


  /**
   * Checks to see if two files are the same.
   *
   * @param f the file to compare to
   * @return true if the files are the same and false if the files are not
   */
  public boolean equals(GG_file  f)
```

```
{
  if(me.compareTo(f.me) == 0)
    return true;
  else
    return false;
}


/**
 * Gives you the name of the file.
 *
 * @return a string that contains the name of the file.
 */
public String getName()
{
  return me.getName();
}


/**
 * Gives you the name of the file.
 *
 * @return a string that contains the absolute path of the file.
 */
public String pathToString()
{
  return me.toString();
}


/**
 * Gives access to the entire content of a file.
 *
 * @return a string containing all the contents of the file.
 */
public String getContent()
{
  String contents = new String();

  try
  {
    in = new BufferedReader(new FileReader(me));
  }
  catch(FileNotFoundException  e)
  {
    System.err.println("Unable to open file "  + me.getAbsolutePath() +
                       " for reading" );
  }

  try
  {
    while (in.ready()) {
      contents += (in.readLine() + "\n");
    }
    in.close();
  }
  catch(IOException  e)
  {
    System.err.println("Unable to read from "  + me.getAbsolutePath());
  }

  return contents;
}
```

```
/**
 * Writes to the file, overwriting previous content.
 *
 * @param s the string to write to the file
 * @return true if write was sucessful, false if failed
 */
public boolean writeContent(String s)
{
  try
  {
    out = new BufferedWriter(new FileWriter(me));
  }
  catch(IOException e)
  {
    System.err.println("Unable to open file " + me.getAbsolutePath() +
                       " for writing");
    return false;
  }

  try
  {
    out.write(s,0,s.length());
    out.flush();
    out.close();
  }
  catch(IOException e)
  {
    System.err.println("Unable to write to " + me.getAbsolutePath());
    return false;
  }

  return true;
}


/**
 * Writes to the file, appending previous content.
 *
 * @param s the string to write to the file
 * @return true if write was sucessful, false if failed
 */
public boolean appendContent(String s)
{
  String contents = new String();

  try
  {
    in = new BufferedReader(new FileReader(me));
  }
  catch(FileNotFoundException e)
  {
    System.err.println("Unable to open file " + me.getAbsolutePath() +
                       " for reading");
  }

  try
  {
    while (in.ready()) {
      contents += (in.readLine() + "\n");
    }
    in.close();
```

```
    }
    catch(IOException  e)
    {
      System.err.println("Unable to read from "  + me.getAbsolutePath());
    }

    contents += s;

    try
    {
      out = new BufferedWriter(new FileWriter(me));
    }
    catch(IOException  e)
    {
      System.err.println("Unable to open file "  + me.getAbsolutePath() +
                         " for writing");
      return false;
    }

    try
    {
      out.write(contents,0,contents.length());
      out.close();
    }
    catch(IOException  e)
    {
      System.err.println("Unable to write to "  + me.getAbsolutePath());
      return false;
    }

    return true;
  }
}
```

```
#!/bin/bash
#Thanks Lukas

tpath=../TestSuiteNew/Syntax/

globallog=${tpath}GGTests.log

output_tex=${tpath}lexer_test.log

rm -f ${tpath}$output_tex
rm -f ${tpath}$globallog

error=0

Check() {
echo "Trying file: $1"
basename=$(echo $1 | sed 's/.gg$//')
reffile=$(echo $1 | sed 's/.gg$/.BASECASE/')

lexeroutputfile=$basename.tree          #where we redirect output of lexer
difffile=$basename.diff                 #see if they differ
echo -n "Parsing test on: $basename..."
echo "Parsing $1" 1>&2
#run java Main <gg file> and output to

echo -n "Here's $1 1>&2"
java "Main3" $1 > $lexeroutputfile 2>&1 || {
        echo "FAILED: gg lexer/parser terminated"
        echo "FAILED: gg lexer/parser terminated" 1>&2
        error=1; return 1
        }
diff -b $reffile $lexeroutputfile > $difffile 2>&1 || {
        echo "FAILED: output mismatch"
        echo "FAILED: output mismatch" 1>&2
        error=1 ; return 1
        }

        #rm $lexeroutputfile $difffile
        echo "file PASSED"
        echo "file PASSED" 1>&2
}

generate_output() {


basename=$(echo $1 | sed 's/.gg$//')
reffile=$(echo $1 | sed 's/.gg$/.BASECASE/')
genfile=${path}$basename.tree
difffile=${path}$basename.diff

echo "GG source file: $1" >> $output_tex

cat $1 $genfile $reffile >> $output_tex

echo "Diff with BASECASE:" >> $output_tex
cat $difffile >>$output_tex

}

for file in ${tpath}Test*.gg
do
        Check $file 2 >> $globallog
done

for file in ${tpath}Test*.gg
do
        echo "Trying to cat: $file"
```

```
        generate_output $file
done

echo "FINAL RESULTS FOR THE TEST RUN:" >> $output_tex
cat $globallog newline >> $output_tex

echo "END OF TEST" >> $output_tex

#do some cleanup
rm ${tpath}*.diff

exit $error
```

```
void main()
{
  int a = 3;
  if(a > 0)
  {
     a = 2;
  }
}
```

Appendix B: Meeting Minutes

PLT Meeting Minutes February 1, 2003

"What are you doing in the command line?"

Data Types

- boolean
- int
- float
- char

Note: type conversion allowed (if done with two lines of code),
int ? float
int ? char

Our Classes

- FTP
- SSH
- HTTP
- Sockets
- File (for uses such as logs)
  - part of FTP class?
- Standard I/O

Java Classes

- String (including type conversion methods)
- Arrays

Data Structures
(none provided)

Control Flow

- if/else
- for
- while
- return
- Method calls

General discussion of Class interpretation

- All files must be in one directory
- Only one main method/directory
- Create 1 java file/Class
- Java file named same as your Class

Discussion for next time: More detail in classes

Tasks for next time: Kierstan: Reference manual skeleton

Jonah: Java networking code
Elizabeth: GUI parameters, absolute positioning possible?
Jake: Begin to outline test suite

PLT Meeting Minutes February 09, 2003

"We give it to you"

GUI Components

- GGFrame
- GGButton (come with action listeners)
- GGTextArea
- GGTextField
- GGTerminalBox
- GGFileTree
- GGLabels
- GGDialogBox
- GGRadioButton
- GGCheckBox

(menus included in version 2.0)

Networking

- FTP
- SSH
- HTTP
- Sockets (character based)

Goal: absolute positioning

Next time

- Control Flow

PLT Meeting Minutes February 16, 2003

"Because when everything else shits the bed, you can still ring the bell!"

General discussion

- Java networking packages and Applets found (specifically for FTP and SSH, respectively)
- Used Reference Manual as outline for discussion (including operators and keywords)

Discussion for next time: More on the reference manual.

Tasks for next time: Kierstan: Continue reference manual skeleton, read next sections of the C and Java reference manuals ? topics for next week, clean up the beginnings of the GG reference manual, send email to Professor Edwards concerning plagiarism.
Jonah: Continue working with Java networking packages found.
Elizabeth: GUI parameters, absolute positioning possible?
Jake: Continue to outline test suite

PLT Meeting Minutes February 24, 2003

"If you need more than 4, go to hell"

Overview: No absolute positioning ? redefine GUI components ? supply predefined panels (makes for easier error checking). Note: These changes will need to be reflected in the revised white paper.

Button Panel
- 1 x 1
- 1, 2, 3, or 4/panel
- labels

Radio Button Panel
- 1 x 1
- 2, 3, or 4/panel
- labels

Check Box Panel
- 1 x 1
- 1, 2, 3, or 4/panel
- labels

File Tree Panel
- 2 x 2
- Name, Date, Size

Text field Panel
- 1 x 1
- 1 or 2/panel
- labels

Text Area Panel
- 1 x 1, 2 x 2, 3 x 2

Label Panel
- 1 x 1, 2 x 1
- up to 3 lines of text

Terminal Box Panel
- 3 x 3

Dialog Box
- OK
- Yes/NO
- Warning

Frame

- 5 x 4, 3 x 3, 2 x 2, 4 x 4

Note: see notes for layouts.

Tasks for next time: Kierstan: Continue to work on reference manual
Jonah: Continue to work on guts
Elizabeth: GUI components skeleton
Jake: Continue to work on test suite

PLT Meeting Minutes February 26, 2003

Overview: Tonight was a short meeting. This is just a reminder to us that we discussed moving ahead with ANTLR and GUI stuff, and splitting our Wednesday meetings into 2 person group meetings to start the actual coding.

Next Time: Monday is the day before the midterm, so we will have a quick meeting to touch base and plan the tasks for the following week.

Tasks for next time:
Kierstan: Continue to work on reference manual and look at ANTLR stuff with Liz
Jonah: Continue to work on guts
Elizabeth: Start looking into ANTLR
Jake: Continue to work on test suite and possibly look at guts stuff with Jonah
All: Study for the midterm!

PLT Meeting Minutes March 3, 2003

Overview: Tonight was a short meeting. This is just a reminder to us that we agreed on getting rid of the GUI aspect of our language. The idea of getting rid of SSH was also tossed into the discussion. Discussed the possibility of overloading, such as a universal print.

Next Time: Discuss the changes more in depth.

PLT Meeting Minutes March 5, 2003

"That's Wild"

General discussion

Now that we have decided to get rid of the GUI, we have also decided to not have object orientation. Instead, we will have a universal "Networking" class with predefined functions.

Beginning ideas of possible functions

- print
- send
- scan
- receive
- setNet(ftp, ssh, Socket)
- send(.., ftp, chdir)
- receive(x, ftp)

We also discussed the possibility of overloading operations (i.e. class + class = array of classes)

We will allow functions to be written

Discussion for next time: Continue to run with the new ideas.

Tasks for next time: Kierstan and Liz: Meet with Edwards to discuss our new ideas since we last met with him.

Jake: New name for scan ☺

PLT Meeting Minutes March 10, 2003

General discussion
This was a very short meeting, we decided to make sure we each have suggestions for our "Networking" class methods, so as to help move along the discussion next week.

Discussion for next time: Try and finalize our new ideas, so that the reference manual rough draft can be complete by the end of Spring break. This includes a complete list of supplied methods.

Tasks for next time: Kierstan and Liz: Edwards was out of town last week, so meet with him Thursday to discuss our new ideas since we last met with him.
All: Come next time with a list of ideas for methods

PLT Meeting Minutes March 13, 2003

"The first Jewish Language" -Jake
"Simon is hot!" -Kierstan

Overview: Great meeting! Hammered out all of the pre-defined functions available in the "Networking" class. Each function is listed below with a brief description,

- *variable = fgetsome(filename);
  - reads line by line

- fprint(filename, append | overwrite, variable);
  - prints to specified file
  - filename = "String" | ptr. to file

- print(variable);
  - prints to terminal
  - if a file, is like cat
  - can concatenate like in Java

- variable = getsome( );
  - will read in then type cast
    - char = ASCII
    - file, string = add up chars

- ftp → int = send(" ", <ftp>);
  - returns an int to reference errors
  - first argument is automatically converted to a string
  - <ftp>
    - cd
    - mkdir
    - rmdir
    - rm
    - put (local)

- socket → int = send(" ");
  - returns an int to reference errors

- ftp → variable = recv(<ftp>);
  - <ftp>
    - ls
    - "String" = filename

- Socket → variable = recv( );

Note: discuss next time what function returns if not noted
*variable = variable of any type

New Keywords:

- cd
- mkdir
- rmdir
- rm
- put
- ls
- Threaded
- append //are these included ad keywords?
- overwrite? //should they be shortened?

General Discussion:

1. All functions can be nested because all functions return something (may need to talk about what some function return next time)
2. file f = “String”, if doesn’t exist, is created
3. everything is public
4. Threaded int myfunc(<arg>);

Discussion for Next Time:

1. Overwrite the + and = operators
2. include ssh and http?
3. creating a server
4. initialization
5. a getime( ) function

Tasks for next time: Kierstan: Complete LRM
Jonah: Work on backend stuff
Elizabeth: learn ANTLR
Jake: Continue to work on test suite

Tasks to be completed by some point:

1. update the white paper
2. meet with him to go over our rough draft, talk about our changes since the meeting with him