Graphr

Michael Cole, Paul Dix, Joseph Kamien, Zhe Chen

December 18, 2007

Contents

Chapter 1

Introduction

Graphr is a language for processing data sets and creating charts and graphs from that processed data. Built into the language are functions and types for specifying different kinds of charts, graphs, histograms, and their source data. The language should be useful in cases where important data must be derived from earlier data. The language is designed to be readable, expandable, and dynamic.

1.1 Motivation

The ability to analyze data and draw conclusions is extremely important. It is difficult to work with large datasets in current spreadsheet software. In cases where multiple functions need to be used on one dataset, the list of dependencies may become unmanageable, and when mistakes are made, debugging is lengthy and difficult. Moreover, since functions must be defined based on locations within the dataset, not based on variables which are meant to be represented by the dataset, it is impossible to reuse functions across multiple datasets. By allowing the programmer to easily define his or her own parser, it will be much simpler to use similar functions across myriad large datasets. By allowing the user to work with functions by variable names, instead of locations within the dataset, debugging will be much simpler.

1.2 Requirements

Graphr is designed to be easily used by scientists and engineers. Therefore, in order to support the principles of multilateral cooperation and peer review, which are hallmarks of the laboratory sciences, the language is designed to be as platform-independent as possible.

Our interpreter is a Java program, so it is necessary to have the Java Runtime Environment (JRE) installed in order to successfully translate and

execute Graphr programs. The latest JRE can be downloaded for free from `java.sun.com`.

1.3 Readability

Graphr's syntax should be a human readable language for describing graphs, and it will be easy for the programmer to specify different mathematical functions. It is important in many real world applications to clarify the path from source data to conclusions drawn from that data. The methods and algorithms for how data files are parsed are understandable, and this is another important consideration for readability.

1.4 Data Types

We plan to make the language dynamically typed in order to help programmers who are not computer scientists understand our language. Therefore, we want to avoid using cryptic types like integer, float, etc which can confuse users who are unfamiliar with these terms. However, there will be a certain number of first class types in the syntax of the language. These include numbers, regular expressions, functions, and graphs. Expandable Graphr includes built in functionality for parsing different source data files like .CSV. However, not all source data file types can be built into the language by default. It will be possible for users of the language to design additional preprocessing modules for parsing file types besides CSV. The language will also have a number of built in chart and graph types. These will define how a graph is drawn and how the data appears on the graph.

1.5 Output

The goal of the language is to quickly parse through data and create charts and graphs, so output is very important. Graphr programs are able to output text to the screen, manipulate text files stored on the user's file system, and output jpeg images of graphs.

Chapter 2

Language Tutorial

2.1 A Simple Example

The first and most simple example of a programming language is almost always the code to produce "Hello World". Here is the Graphr implementation off the simple program to produce "Hello World" world to the console:

```
puts "Hello World";
```

As you can see, Graphr can implement this basic program in a single line of code! Easy isn't it? The function **puts "Hello World"** prints the string "Hello World" , which is enclosed in quotes, out to the command line. The **;** indicates the end of the statement, and is mandatory at the end of every Graphr statement.

Now let's look at an example that actually shows some useful properties of the Graphr language:

```
def life(arg1, arg2){
     t = true;
     if (t){
          puts arg1;
          puts arg2;
          return 4*10+2;
     }else {
          return "There is no purpose in life";
     }
}
a = "The purpose";
b = life(a, "of life is ");
puts b;
```

In this code snipped, we define a function called life with 2 arguments with the statement def **life(arg1, arg2)**. The def is needed when a function is first

defined. The body of the function is enclosed in { and }. The function is called near the end with **life(a, "of life is")** with the two arguments being **a** and **"of life is "**. As you can see variables are assigned a value with the = operator, and all variables are dynamically typed, so they do not need to be declared before use. Notice that **t** is assigned to a Boolean, while **a** is assigned to a string, and **b** is assigned to the return value of the function life.

Inside the function life, there is an if-else statement checking the value of **t**, since **t** is true, the first block of the if-else statement executes, printing out **arg1** and **arg2**, then returning the mathematical expression 4*10+2. The output of this code is:

```
The purpose
of life is
42
```

2.2 Compiling and Running Graphr Code

There are two ways to run Graphr code. The first and main method is to write the code into a .gr file, for example, code.gr, and then running the Graphr interpreter to interpret the .gr file. To invoke the Graphr interpreter on the .gr file, use the following command:

```
java -jar Graphr.jar filename.gr
```

The alternative method is to pass in the code as an argument to invoking the interpreter. For example, if you want to run the 'Hello World' code from the very first example, we invoke as follows:

```
java -jar Graphr.jar -e "puts \"Hello World\";"
```

The -e argument tells the interpreter to interpret the last argument instead of reading it from a file. The code should be encapsulated in quotes. You must be careful, because if you write a quote within your code, the interpreter would think that it had reached the end of the code. So, whenever you write a a quote, you must escape it by placing a \ immediately in front of the quote.

2.3 More Examples

```
def factorial(arg1){
     prod = 1;
     for (a=1; a <= arg1; a+=1){
          prod *= a;
     }
     return prod;
}
def getMean(array){
```

```
        mean = 0;
        size = 0;
        foreach(i in array){
            mean += i;
            size += 1;
        }
        return mean/size;
}
puts factorial(5);
a = [4, 7, 12, 3, 8];
puts getMean(a);
```

In this program, we define a factorial function and a mean function. The factorial function takes in a number and calculates its factorial using a for loop. **An array is declared with the statement a = [4, 7, 12, 3, 8]**. In getMean, a foreach loop is used to grab every value inside the array and add it to mean. Incrementing by one is done by the += 1 assignment operator. This code snippet outputs the following:

```
120
6.8
```

Next, let's look at some library functions we can make use of:

```
data = read_file("data.csv");
puts data;
puts substring("This is not a long string", 0, 8);
puts length("This is not a long string");
puts split("1, 3, 3, 7", ",");

foreach( line in data){
      puts line;
}

writedata = ["35,22", "72,34", "100,2"];
puts write_file("write.csv", writedata, "w");
```

The function **read_file("data.csv")** returns an Array that has each line of the input file data.csv as a String inside the Array. The function **substring("This is not a long string", 0, 8)** takes "This is not a long string" and creates a new String from characters 0-8 of the original String. The function **length("This is not a String")** returns the length of a String, and finally, **split("1, 3, 3, 7")** returns an Array of Strings with each String being a substring of the original String that was split using the token ",". The function **write_file("write.csv", writedata, "w")** takes the Array of Strings writedata and writes it to the file "write.csv".

The output of this code is:

```
[ 1,10, 2,20, 3,15 ]
This is
25
[ 1,  3,  3,  7 ]
1,10
2,20
3,15
true
```

The input file data.csv:

```
1,10
2,20
3,15
```

The output file write.csv:

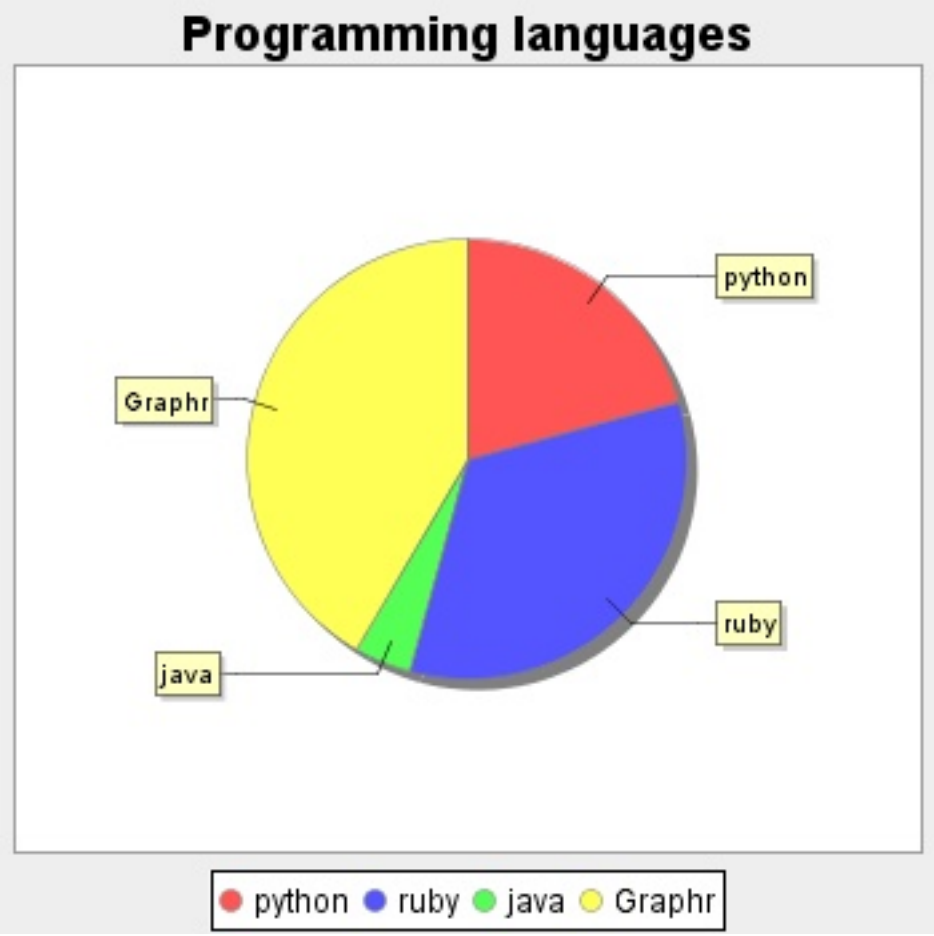
```
35,22
72,34
100,2
```

Finally, let's take a look at how to create a graph:

```
data = {
     "Graphr" => 10,
     "ruby" => 8,
     "python" => 5,
     "java" => 1
};
args = {
     "title" => "Programming languages",
     "type" => "createPieChart",
     "xLabel" => "ignored",
     "yLabel" => "ignored",
     "data" => data,
     "file" => "test_pie.jpg",
     "legend" => true,
     "width" => 350,
     "height" => 350
};
puts data["Graphr"];
chart(args);
```

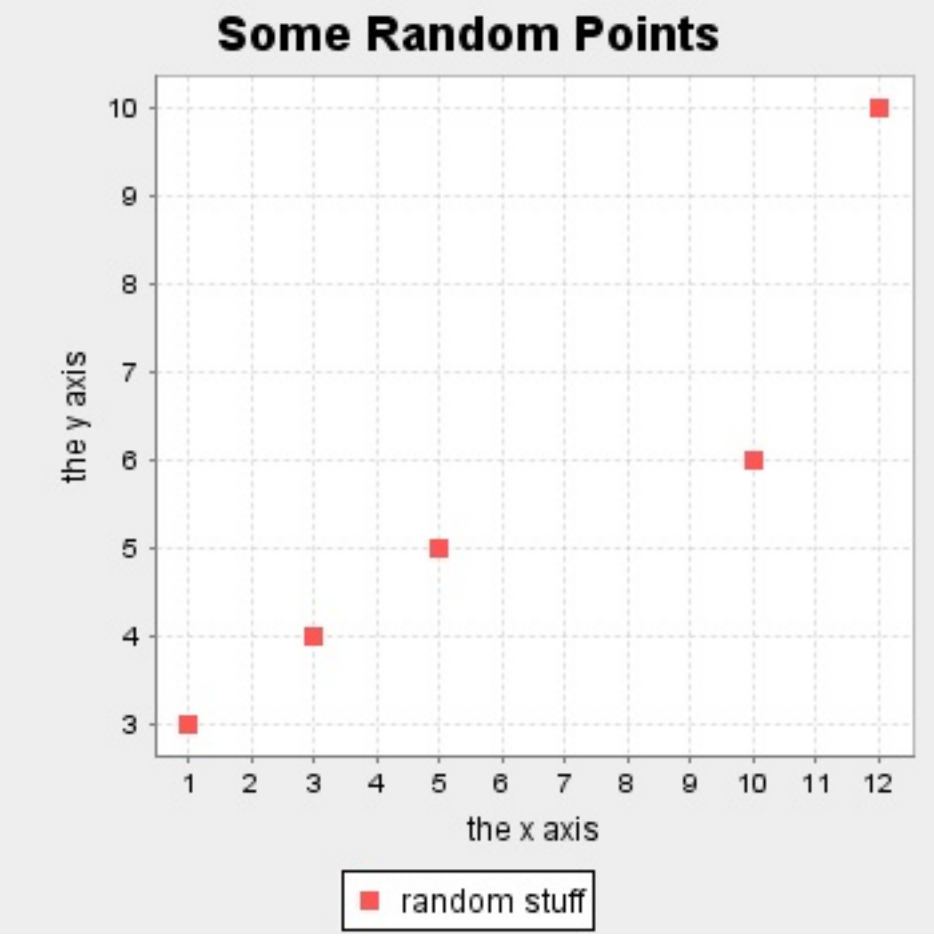
Here we see how Associative Arrays in Graphr are created. In the **data** Associative Array, the numbers 10, 8, 5, and 1 are assigned to the key strings "Graphr", "ruby", "python", and "java". To access a value based on a key in an Associative Array, we use grab the value just like we would in a normal array,

except instead of the index, we input the key, like **data["Graphr"]**. The call to **chart(args)** will draw a jpeg file with the name test_pie.jpg based on the data and values inside the **args** Associative Array.




```
data = [[1, 3], [3, 4], [5, 5], [10, 6], [12, 10]];
graph({
     "title" => "Some Random Points",
     "type" => "createScatterPlot",
     "xLabel" => "the x axis",
     "yLabel" => "the y axis",
     "legendLabel" => "random stuff",
     "data" => data,
     "file" => "test_scatter.jpg",
     "legend" => true,
     "width" => 350,
     "height" => 350
});
```

When the graph requires both an x and y axis, we use graph() instead of chart(). Here, the data is a 2-D array, initialized by initializing an array with each element as another array initialization. Also, this time we are creating the Associative Array as the argument to the function graph(). The output jpg is the following:



Chapter 3

Language Reference Manual

3.1 Language

3.1.1 Characters

3.1.1.1 Escape Character

In order to insert a quote into a string, it is necessary to "escape" the quote by entering \" instead of simply ".

3.1.1.2 Comments

Comments in source code are ignored by the compiler. In-line comments start with /* and continue until the first */ is encountered (even if the first */ is on a subsequent line). Single-line comments begin with // and continue until the end of the line.

3.1.2 Identifiers

3.1.2.1 Keywords

The following keywords are reserved and can not be used.

for	if	nil	else
foreach	while	return	def
true	false	in	or
and	puts	read_file	write_file
include			

3.1.2.2 Variable Declaration

Variable names start with a character a-z or A-Z then continue with zero or more of the following characters a-z, A-Z, 0-9, or underscore _. Variable names are case sensitive.

Variables in Graphr are dynamically typed.

3.1.2.3 Numbers

The underlying structure storing numbers has the precision of a double in C. Here are the number of bits and the possible values a number can store.

64 bits 2.2250738585072014 * (10^-308) to 1.7976931348623157 * (10^+308)

Numbers can be assigned to variables using the following syntax.

```
some_integer = 12;
some_decimal = 3.2;
```

Numbers must start with a digit, a number sign, or a decimal. Numbers that start with a digit can be followed by any number of digits and optional decimal point and zero or more digits afterward. Numbers that start with a decimal must have one or more digits following the decimal.

3.1.2.4 Arrays

Arrays are single or multidimensional matrices. They are defined by assigning a set of values separated by commas and bracketed by [] characters to a variable. The function "length," originally designed to return the length of strings and described in section 3.2.4.1, has been overloaded so that, if given an array, it will return the length of that array. Examples:

```
my_array = [2, 4, 2];
my_multi_dimensional_array = [[2, 3, 1], [4, 5]];
my_array_holding_different_types = ["hello world", 23.34, some_variable];
```

Memory for arrays is allocated at run time. They should start with a minimal number of elements and increase in size dynamically. Arrays are zero indexed and elements within an array can be referenced by including an expression that evaluates to a number within the postfix brackets on an array name. Examples:

```
puts my_array[2];
puts my_array[some_function_that_returns_a_number()];
```

3.1.2.5 Associative Arrays

Associative arrays are arrays that have their elements dereferenced by a hashed value rather than an integer. They are defined by assigning a set of key, value pairs separated by commas and enclosed in curly braces. The key must be a string, the value can be any type of Graphr data (including strings, numbers, arrays, and even other associative arrays). Each key value pair is specified with the key on the left and the value on the right with the two separated by =>. Examples:

string_assoc_array = {“a string” => “some value”, “another string” => “another value”};
num_assoc_array = {“one string” => 1, “two string” => 2};
mixed_assoc_array = {“one string” => “one value”, “two num” => 2};

The program can access individual values stored in an associative array using the same syntax as with a regular array.

Example:

puts string_assoc_array[“a string”]; //prints “some value”

3.1.2.6 Strings

Strings are an array of characters surrounded by double quotes. Examples:

my_string = “hello world”;
my_string = “\tThis is indented”;

3.1.3 Scope

Variable scope is static with no global scope. Outside of functions, there is a main scope. Variables declared in this scope can only be used and affected by expressions in this scope. Functions can only use variables declared within themselves or those passed to it as arguments. Variables used within a function cannot be accessed outside that function, although the function can return the value of a variable. The result is that in our language, calling functions has no “side effects.”

For example, f(1) + f(2) will always equal f(2) + f(1) for any Graphr function f.

3.1.4 Functions

3.1.4.1 Function Definition

A function is declared using the def keyword. They are declared like this:

def function_name(parameter-list, ...) { body... }
Function_name is the name of the function. Allowable function names are the same as those for variables.

Parameter-list is the list of parameters that the function takes separated by commas. If no parameters are given the parameter list should be defined with an empty set of parenthesis.

All functions must return a value. The value they return is specified by a return statement within the function body. Examples:

```
return 3;
return my_associative_array;
return "some string";
```

All functions must be declared at the top level and will be accessible as if they were global variables (even though Graphr itself does not actually have global variables). The return statement must be the last line in a function, and there cannot be more than one return statement in the same function. If a return statement is not provided, the function will automatically return a Boolean with the value "true."

3.1.4.2 Program Startup

The entire program is recognized as one expression. It can be stored in a file with the .gr extension and or it can be enclosed in quotes and run via the command line.

3.1.5 Operators

3.1.5.1 Prefix

Prefix operators are operators that are attached to the beginning of an expression.

!operand

Returns the logical NOT operation on the operand. A true operand returns false, a false operand returns true.

3.1.5.2 Mathematical

There are several mathematical operators which return the result defined for each. If a string is provided for any of the expressions in a mathematical operation, Graphr will attempt to parse that string as a number.

For example, "5" + 5 would return a number with the value 10.

expression1+expression2

The result of this is the sum of the two expressions.

expression1-expression2

The result of this is the difference of the two expressions.

expression1*expression2

The result of this is the product of two expressions.

expression1/expression2

The result of this is the quotient of the two expressions.

expression1%expression2

The result of this is the value of the remainder after dividing expression1 by expression2. Also called the modulo operator.

3.1.5.3 Boolean

The Boolean operators return either **true** or **false**. Everything which does not evaluate to either **false** or **nil** is considered **true**.

expression1&&expression2

Returns the logical AND operation of expression1 and expression2. Can also be written:

> expression1 **and** expression2

expression1||expression2

Returns the logical OR operation of expression1 and expression2. Can also be written:

> expression1 **or** expression2

expression1<expression2

Returns true if expression1 is less than expression2, otherwise the result is false.

expression1>expression2

Returns true if expression1 is greater than expression2, otherwise the result is false.

expression1<=expression2

Returns true if expression1 is less than or equal to expression2, otherwise the result is false.

expression1>=expression2

Returns true if expression1 is greater than or equal to expression2, otherwise the result is false.

expression1==expression2

Returns true if expression1 is equal to expression2, otherwise the result is false. String equality is based on case sensitive lexical comparison.

expression1!=expression2

Returns true if expression1 is not equal to expression2, otherwise the result is false.

3.1.5.4 Assignment

An assignment operator stores the value of the right expression into the left expression. All assignment operators return the value of the right expression.

expression1=expression2

The value of expression2 is stored in expression1.

expression1*=expression2

The value of expression1 times expression2 is stored in expression1.

expression1/=expression2

The value of expression1 divided by expression2 is stored in expression1.

expression1%=expression2

The value of the remainder of expression1 divided by expression2 is stored in expression1.

expression1+=expression2

The value of expression1 plus expression2 is stored in expression1.

expression1-=expression2

The value of expression1 minus expression2 is stored in expression1.

3.1.5.5 Precedence

The operators have a set order of precedence. Items inside parenthesis are evaluated first and have the highest precedence. The following chart shows the order of precedence with the items at the top having highest precedence.

Operator	Name
!	Logical NOT
* / %	Multiplicative operators
+ -	Additive operators
< > <= >=	Inequality comparators
== !=	Equality comparators
&& and	Logical AND
|| or	Logical OR
+=, -=, ...	Assignment

3.1.6 Conditional Expressions

3.1.6.1 if

The if statement allows for control-flow manipulation. It consists of the reserved word "if" followed by an expression inside parenthesis, followed by a block of code in braces. If the expression within the parenthesis evaluates to true, then the block of code within braces will be executed, if not then this block of code will be ignored. An else statement may be placed immediately following the end of this block of code. The else statement consists of the reserved word "else" followed by a block of code in braces. If the expression does not evaluate to true, and an else statement is provided, then the block of code following the "else" will be executed.

Syntax:

```
if( expression ) {statement1; statement2; statement3;}
or
if( expression ) {statement1; statement2; statement3;}
else {statement4; statement5; statement6;}
```

3.1.6.2 while

The while statement provides an iterative loop.

Syntax:

```
while( expression1 ) {statement1; statement2; statement3;}
```

The block of code {statement1; statement2; statement3;} is executed repeatedly as long as expression1 is true. The test on expression1 takes place before each execution of {statement1; statement2; statement3;}.

3.1.6.3 for

The for statement allows for a controlled loop.

Syntax:

```
for( expression1 ; expression2 ; expression3 ) expression4...
```

expression1 is evaluated before the first iteration. After each iteration, expression3 is evaluated. expression1, expression2 or expression3 can be replaced with nil. If expression2 is nil, it is assumed to be false. expression4 is executed repeatedly until the value of expression2 is false. The test on expression2 occurs before each execution of expression4.

3.1.6.4 foreach

The foreach statement allows for iterating through an array. If the array is associative, it allows for iterating through the keys of that associative array.

Syntax:

```
foreach(element in myarray) { ... }
foreach(element in expression) { ... }
foreach(key in associative_array){ ... }
```

In the above example element represents a new variable that will be in scope of the foreach block. "in" is a keyword separating the new variable and the expression or variable on the right hand side.

3.2 Library

Libraries consist of functions to expand the utility of the program. We created libraries to enable input and output via files and graphical manipulation.

3.2.1 Input Output Library

The Input Output library contains the functions necessary for reading input and printing output. This gives the user the ability to take in data from sources outside of the source code, such as the standard input or other files. This also allows the user to write data to files.

3.2.1.1 read_file

Declaration:

def readfile(*filename*);

Reads from filename and returns the entire file in an array with each line of the file as a string in the array if successful. If the file operation is not successful, then it returns an empty array.

3.2.1.2 write_file

Declaration:

def write_filef(filename, data, writehandle);

Returns true if writing the file is successful, otherwise, nil. The data argument consists of an Array of Strings, with each String printed onto a new line. The writehandle is a String literal that accepts 'a' to append. If the 'a' is not given, then it will overwrite any existing file which has the same name.

3.2.1.3 include

Declaration:

def include(*filename*);

Returns true if *filename* is successfully found. *Filename* should be a Graphr header file or a Graphr source code. Acts as if the entire content of *filename* is copied into the code with brackets around it. If *filename* is not found, returns an empty array. Includes should be placed at the beginning of a Graphr program. Programs which include other programs cannot themselves be included in any other program.

3.2.2 Graphics Library

The Graphics Library takes care of creating graphs from a set of data along with various user inputs for the different components of the graph. There are 2 graph generating functions: chart and graph.

3.2.2.1 Chart

Declaration:

```
def chart(args);
```

The function chart() is used to create graphs that only require a one dimensional set of data, such as pie graphs, histograms, etc. It takes an Associative Array args with the following required attributes:

```
args = {
     "title" => "Name of the Chart Title",
     "type" => "Type of the Chart you want to create",
     "xLabel" => "Label of the X-Axis",
     "yLabel" => "Label of the Y-Axis",
     "data" => AssociativeArray, //an Associative Array
     "file" => "name of the picture output",
     "legend" => Boolean, //of whether to display the legend
     "width" => Number, //indicating image width
     "height" => Number, //indicating image height
}
```

The following chart types are accepted by the chart function as value of the "type" key:

createAreaChart

createBarChart

createBarChart3D

createLineChart

createLineChart3D

createPieChart

createRingChart

createStackedAreaChart

createStackedBarChart

createStackedBarChart3D

createWaterfallChart

3.2.2.2 Graph

Declaration:

def graph(args);

The function graph() creates graphs that require a 2 dimensional set of data, such as scatterplots. It takes an Associative Array args with the following required attributes:

```
args = {
     "title" => "Name of the Chart Title",
     "type" => "Type of the Chart you want to create",
     "xLabel" => "Label of the X-Axis",
     "yLabel" => "Label of the Y-Axis",
     "legendLabel" => "Label of the legend",
     "data" => 2DArray, // A 2-dimensional array containing ordered pairs.
     "file" => "name of the picture output",
     "legend" => Boolean, //of whether to display the legend
     "width" => Number, //indicating image width
     "height" => Number, //indicating image height
}
```

Not charts types use all of the arguments, but you are required to input them regardless. The following chart types are accepted by the chart function as value of the "type" key:

createScatterPlot

createTimeSeriesChart

createXYAreaChart

createXYLineChart

createXYStepAreaChart

createXYStepChart

3.2.3 String Library

The String Library consists of simple functions for parsing Strings which makes it easier to parse through text read in from files.

3.2.3.1 Split

Declaration:

```
def split(String, token);
```

Breaks up String into an Array of Strings splitting based on the argument token, which is also a String. For example, if token is ",", if String was "35, 36, 45", an Array of ["35", "36", "45"] would be returned.

3.2.3.2 Substring

Declaration:

```
def substring(String, start, end);
```

Returns a String that consists of the characters from start to end of the original String, where start and end are integers.

3.2.4 Utility Functions

3.2.4.1 Length

Declaration:

```
def length(String);
def length(Array);
```

Returns the length of the String. This has been overloaded so that if called on an array, it will return the number of elements in that array. If called on an associative array, it returns the number of keys in the array.

3.2.4.2 Contains

Declaration:

```
def contains(Associative_Array, key_string);
```

Returns a Boolean indicating whether or not the associative array contains the key string as one of its keys.

3.2.4.3 Union

The + sign has been overloaded so that if used on arrays, it will return a new array in which the second array is the final element of the first array. This also works for assignment.

Example:

```
a = [2,3];
b = [456,400];
a += b;
// a now equals [2, 3, [234, 12]]
```

If used on an associative array, it will return an array which contains the key/value combinations from each array. If two identical keys point to unique elements, the key/value combination from the second associative array will be used, and the key/value combination from the first array will be ignored.

Chapter 4

Project Plan

4.1 Design Process

4.1.1 Planning & Specification

For the planning and specification stages in the project, we did almost all of our work at weekly group meetings at which every team member was present. We brainstormed different ideas, and weighed the advantages and disadvantages of each. As a group, we crafted the initial design plan, our proposal, and our language reference manual.

4.1.2 Development

4.1.2.1 Lexer & Parser

When initially developing our lexer and parser, we did all of our programming at meetings with every team member present. We would alternate with one team member typing our grammar, one looking over his shoulder to assist in immediate debugging, one working on future grammar additions on paper, and one testing the grammar. This methodology had numerous advantages.

1. It made sure that every team member had an accurate view of the overall code.
2. It eliminated the possibility of conflicts caused by multiple group members working on the same file at the same time.
3. Group members could work without worrying that the work they were doing had already been done or would be incompatible with work done by other members.

However, as the length of the parser grew, it became more difficult to find lengthy blocks of time during which each group member could be present. We finished the parser by working on it either as individuals or in pairs.

4.1.2.2 Static Semantic Analysis

We began working on this part of the project as individuals. We noticed over time that this portion of the project was extremely difficult for one person to do individually, and we resumed our group work strategy. We finished this part of the project at meetings at which every group member was present.

4.2 Programming Style Guide

4.2.1 Version Control

As a group we use Subversion, provided by Google Code (`http://code.google.com/`), to automate version control. Each team member will regularly submit updates to whichever part of the project he his working on, and must appropriately mark changes in the log. A team member must always perform an update (making sure that he has the latest version) before committing any code to the repository.

4.2.2 Documentation

We will appropriately comment every element of the code, using Javadoc if possible.

4.2.3 Naming

Code will be modularized and divided into separate files whenever possible to allow multiple people to work on the same aspect of the project at once. Functions and variables will be given names that allow other group members to intuit their meanings.

4.2.4 Braces

We utilize the brace style used in *The C Programming Language* by Kernighan and Ritchie. When describing control flow, the first opening brace will be placed on the same line as the control statement. The closing brace will be written at the same indent level as the original opening control-flow statement.

Example: as opposed to the lines

```
for(int i = 0; i < 10; i++)
{
     print i;
}
```

or

```
for(int i = 0; i < 10; i++){
     print i;
     }
```

we would write

```
for(int i = 0; i < 10; i++){
     print i;
}
```

4.2.5 Vertical Display

We will use spacing to make sure that, where appropriate, like values appear as a vertical chart.

Example:

```
if (c.containsKey("add_circle"))                    init_circles();
if (c.containsKey("add_rectangle"))                 init_rectangles();
if (c.containsKey("add_line"))                      init_lines();
```

4.3 Project Timeline

9/13/07

Began to brainstorm ideas for languages; agreed to meet at later time and come to final decision.

9/17/07

Decided to create a language for making charts and graphs, began working on proposal.

9/25/07

Submitted completed project proposal. Planned a meeting with TA to receive feedback.

10/13/07

Modified our project based on feedback, largely altering the syntax. Set up Subversion with Google Code (discussed further in section 4.2.1). Began working on language reference manual and the initial ANTLR grammar.

10/18/07

Submitted completed LRM.

10/24/07

Finished ANTLR grammar. Began working on graphics package and static semantic analysis.

11/4/07

Finished rough draft of graphics package.

12/12/07

Finished integrating the charts & graphs package with the rest of our code.

12/18/07

Finished project (static semantic analysis & integration with graphics).

4.4 Member Roles

As explained in the aforementioned section, every member is responsible for participating in every aspect of the project. However, we do have official distinctions based on specific portions of the project.

Team Member	Official Role
Michael Cole	Grammar, parser, and lexer
Paul Dix	Static semantic analysis
Joseph Kamien	Testing and documentation
Zhe Chen	Integration with Java graphics

4.5 Software Development Environment

4.5.1 Languages

We use ANTLR v3 (`http://www.antlr.org/`) for generating our lexer and parser. We use Java v1.5 (`http://java.sun.com`) for our static semantic analysis, and we use Ruby v1.8.6 (`http://www.ruby-lang.org/`) for regression testing.

4.5.2 Tools

We use the Antlrworks v1.1.4 integrated development environment to assist in generating our lexer and parser. As mentioned in section 4.2.1, we utilized Subversion provided by Google Code to automatically manage version control. When working with Java, we used a combination of simple text editors (such as WordPad and TextMate) and Eclipse (a more sophisticated IDE available at `http://www.eclipse.org/`).

4.6 Project Log

We logged our project using the built-in log features provided by Subversion. Since we were using Google Code, we were also able to use its "issues" feature

to keep track of small changes within our project. Since we did most of our coding as a group, it was rarely necessary to use this latter feature, since we could just talk to each other as issues arose. For the complete subversion log, and the complete Google Code "Issues" log, please see section A on page 44.

Chapter 5

Architectural Design

5.1 Major Components

The major components of our translator are described by the following block diagram:

For the user's convenience, we have constructed a tool which performs the above steps and then executes the Java program using the Java Runtime Environment (described as a requirement in section 1.2) installed on the user's computer.

5.2 Interface Description

Preprocessing Before the Graphr file is seen by the lexer, the Graphr.java program performs preprocessing, handling include statements as necessary.

Lexer This part scans the .gr file and identifies different tokens.

Parser This part uses top-down predictive parsing to tell whether the tokens are actually organized in the form of a valid program. The interface between the parser and the lexer (the exact format in which tokens are stored) is handled automatically by the ANTLR tool.

Tree-Walker This part received the abstract syntax tree represented by a stream of tokens. ANTLR v2 allows the program to jump to different points in the tree, but ANTLR v3 only supports iterating through the stream and rewinding the stream (for more information on the software

we used, see section 4.5). This would not be an issue if our code executed everything sequentially, but for loops and control flow operations, it is necessary to jump to different parts of the tree. The tree-walker accomplishes this by marking certain points in the stream, and rewinding or fast-forwarding to marked points in the stream as necessary.

Execution This interface sends Java code directly to the Java Runtime Environment (JRE). We require the user to have the JRE installed on the machine on which he or she runs Graphr code. For more information, see section 1.2.

Chapter 6

Test Plan

6.1 Test Programs

We used several programs to test the functionality of our project. Here is a sample of some programs we used.

6.1.1 Employee of the Month Program

This program is designed to read from a data file, calculate which employee had the most sales, and display the resulting data in a graph. It requires the file "salesdata.csv" to be in the same directory as the Graphr interpreter. Here are the sample contents of "salesdata.csv":

```
April
Bill,30
Ted,15
Bob,20
```

Here is the code for the program itself:

```
//reads data from file
filename = "salesdata.csv";
data = read_file(filename);
len = length(data);
puts "Number of employees: " + (len - 1);

//checks file read successfully
if(len > 0){
     //parses file
     month = data[0];
     puts "Calculating graph for month of " + month;
     chartdata = {};
```

```
     maxsales = 0;
     sales = 0;
     person = "";
     for(i = 1; i < len; i+=1){
          linestr = data[i];
          line = split(linestr,",");
          key = line[0];
          sales = line[1];
          chartdata[key] = sales;
          if(sales > maxsales){
               person = key;
               maxsales = sales;
          }
     }
     puts chartdata;
     puts "Employee of the Month: " + person +
          " with " + maxsales + " sales!";
     //makes chart
     a = 3;
     chart({
          "title" => "Congratulations " + person,
          "type" => "createBarChart",
          "xLabel" => "Employee",
          "yLabel" => "Sales",
          "legendLabel" => "",
          "data" => chartdata,
          "file" => "bar.jpg",
          "legend" => false,
          "width" => 350,
          "height" => 350
     });
     }
else{
     puts "Error reading from file: " + filename;
}
```

Here is the resulting standard output:
Number of employees: 3
Calculating graph for month of April
[Bill => 30, Bob => 20, Ted => 15]
Employee of the Month: Bill with 30 sales!
Here is the resulting graph created with this dataset:

6.1.2 The Standard Graph Program

One of the main advantages to a language like Graphr is that it has the ability to include other Graphr programs. This encourages collaboration by allowing shared functions to be declared in one file, and then used in many programs. This example program shows how the include function can be utilized to create standard graphs across organizations.

The following Graphr file should be in the same directory as the Graphr interpreter and named "graphlib.gr":

```
def scatgraph(d, name){
     graph({
     "title" => "Some Random Points",
     "type" => "createScatterPlot",
     "xLabel" => "the x axis",
     "yLabel" => "the y axis",
```

```
     "legendLabel" => "random stuff",
     "data" => d,
     "file" => name,
     "legend" => true,
     "width" => 350,
     "height" => 350
     } );
}
def linegraph(d, name){
     graph({
     "title" => "Some Random Points",
     "type" => "createXYLineChart",
     "xLabel" => "the x axis",
     "yLabel" => "the y axis",
     "legendLabel" => "random stuff",
     "data" => d,
     "file" => name,
     "legend" => true,
     "width" => 350,
     "height" => 350
     } );
}
```

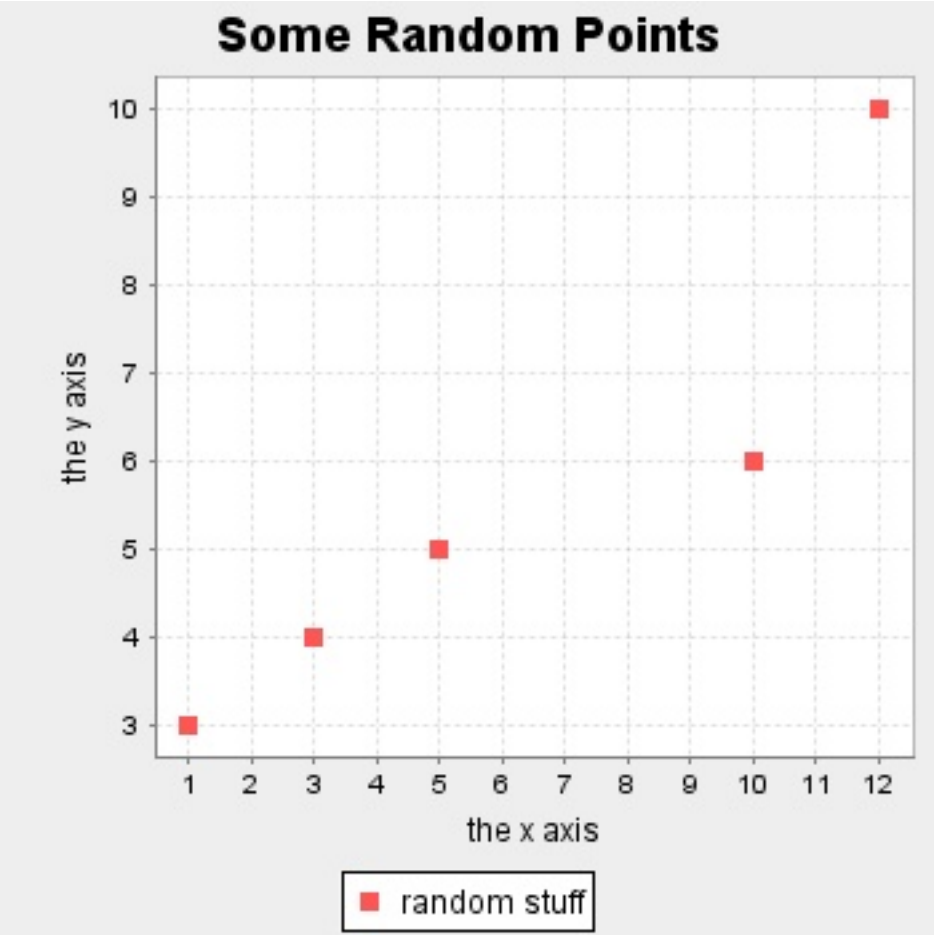
Any other Graphr program can include this program, and then call upon either function. The result is that an organization can create the "graphlib.gr" file and allow all of its members to make graphs which have the same formatting. Here is a sample program which utilizes this feature:

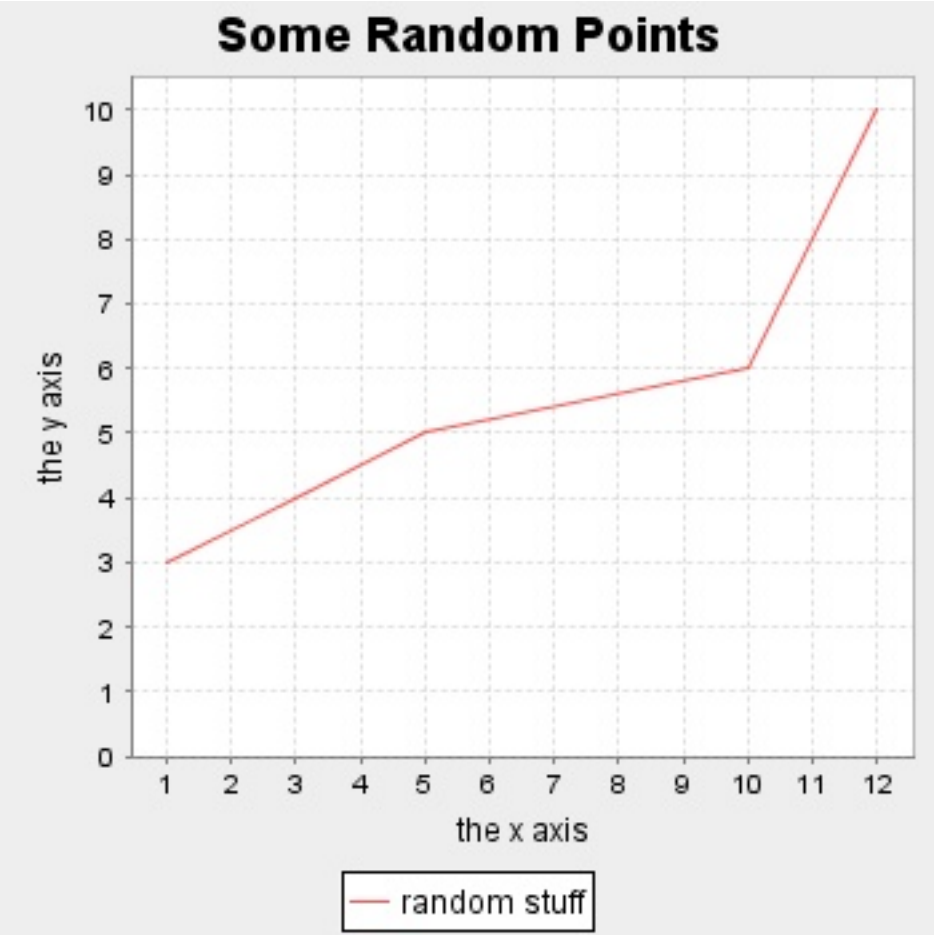
```
include "graphlib.gr";
data = [[1, 3], [3, 4], [5, 5], [10, 6], [12, 10]];
linegraph(data, "my_linegraph.jpg");
scatgraph(data, "my_scatgraph.jpg");
data = [];
foreach(line in read_file("data.csv")) {
     data += split(line, ",");
}
puts "Here is the data:";
puts data;
linegraph(data, "graph_from_file.jpg");
```

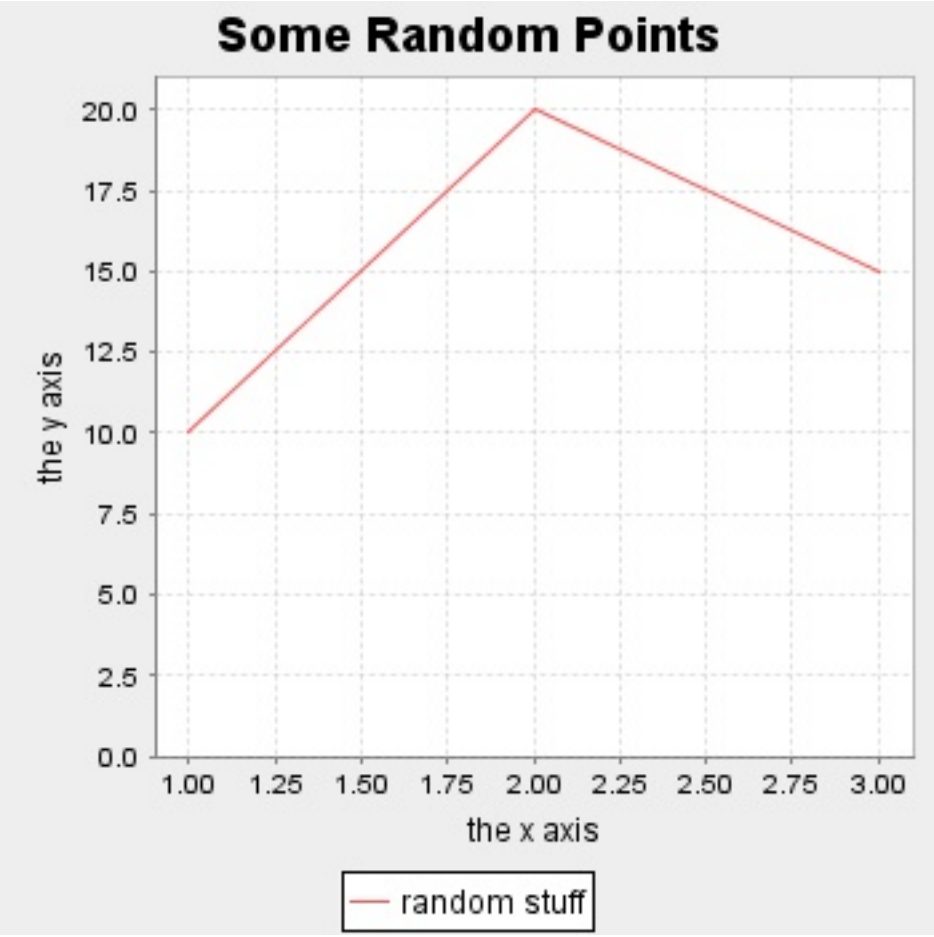
Here is the standard output produced:

```
Here is the data:
[ [ 1, 10 ], [ 2, 20 ], [ 3, 15 ] ]
```

Here are the resulting graphs which are produced:


Some Random Points
the y axis
10
9
8
7
6
5
4
3
1
2
3
4
5
6
7
8
9
10
11
12
the x axis
random stuff


Some Random Points
the y axis
0
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
11
12
the x axis
random stuff



6.2 Test Suites

We used the following test suites to test our interpreter:

6.2.1 String Tests

```
'puts length("hello");',
'foreach (sub in split("this, is, a, test string", ", "))
{puts sub;}',
'a = "this, is, a, test string";
foreach (sub in split(a, ", ")) {puts sub;}',
'puts substring("hello world", 0, 3);',
'a = "hello, world"; puts substring(a, 5, length(a));'
```

6.2.2 File I/O Tests

```
"a = read_file(\"test.txt\"); puts a;",
"foreach (line in read_file(\"test.txt\")) {puts line;}",
"a = read_file(\"test.txt\"); a[2] = \"modified line\";
puts write_file(\"modified.txt\", a, \"w\");"
```

6.2.3 Associative Array Tests

```
'a = {}; puts a;',
"a = {\"asdf\" => 3, \"foo\" => \"bar\"}; puts a;",
"a = {\"asdf\" => 3, \"foo\" => \"bar\"}; puts a[\"foo\"];",
"a = {\"asdf\" => 3, \"foo\" => \"bar\"}; a[\"foo\"] = \"hello world\";
puts a[\"foo\"];",
"a = \"hello\"; b = {\"foo\" => 3, \"bar\" => a}; puts b;"
```

6.2.4 Control Flow Tests

```
"a = true; if (a) {puts \"a was true\";}",
"a = false; if (a) {puts \"we shouldn't get here\";}
else {puts \"a was false\";}"
```

6.2.5 Array Tests

```
'a = []; puts a;',
"puts [2, 1, 3, 9, 10];",
"a = [2, 1, 3]; puts a;",
"a = [\"test\", \"stuff\"]; puts a;",
"a = [\"test\", \"stuff\"]; puts a[1];",
"a = [\"test\", \"stuff\"]; a[0] = \"hello world\"; puts a[0];",
'a = ["foo", "bar"]; a += [3, 2]; puts a;'
```

6.2.6 Foreach Tests

```
'foreach (i in ["hello", "world"]) {puts i;} puts "done";'
```

6.2.7 While Loop Tests

```
"i = 0; while (i < 3) {puts i; i += 1;}",
"t = true; while (t) {puts t; t = false;}"
```

6.2.8 For Loop Tests

```
"i = 3; for (i; i < 5; i += 1) { puts i; puts \"hello\";}",
"for (i = 2; i < 5; i += 1)
{puts i; puts \"hi from second\";}"
```

6.2.9 Function Tests

```
'def myfun() {a = "test"; puts a;} myfun();',
"def myfun(arg1, arg2) {puts arg1; puts arg2;}
myfun(\"hello\", \"world\");",
"def myfun()
{puts \"hello from function with no arguments\";}
myfun();",
"def myfun(arg) {puts arg;}
myfun(\"a function with one argument\");",
'def myfun() {puts "in function"; return 3;
puts "this should NOT output";}
a = myfun();
puts "returned value: " + a;',
'def myfun()
{puts "in function";
return 3;
puts "this should NOT output";}
puts myfun() + 3;',
'def myfun(arg) {puts "in function";
return arg + 2;
puts "this should not print!";}
puts "returned value: " + myfun(2);'
```

6.2.10 Logical Tests

```
"foo = true; bar = false; test = foo or bar;    puts test;",
"foo = true; bar = false; test = foo || bar;     puts test;",
"foo = true; bar = true; test = foo and bar;     puts test;",
"foo = true; bar = true; test = foo && bar;      puts test;",
"foo = 2; bar = 3; test = foo < bar;            puts test;",
"foo = 2; bar = 2; test = foo <= bar;            puts test;",
"foo = 3; bar = 2; test = foo > bar;            puts test;",
"foo = 3; bar = 3; test = foo >= bar;            puts test;",
"foo = 4; bar = 4; test = foo == bar;            puts test;",
"foo = 4; bar = 4; test = foo != bar;            puts test;",
'puts !true;',   'a = false; puts !a;',
't = 1; if (t == 1) {puts "hi";}'
```

6.2.11 Assignment Tests

```
"test = 234;             puts test;",
"test = 1; test += 2;    puts test;",
"test = 2; test -= 1;    puts test;",
"test = 6; test /= 2;    puts test;",
"test = 8; test *= 2;    puts test;",
```

```
"test = \"some string\"; puts test;",
"test = 2 + 3;           puts test;"
```

6.2.12 Math Tests

```
"a = 3 + 4 + 7; puts a;",   "a = 3 * 5; puts a;",
"a = (3 + 4) * 5; puts a;",   'puts 3 % 2;',
'puts "hello" % 5;',   'puts "4" % 2;',
'puts 5 % "3";'
```

6.2.13 Variable Tests

```
"a = 7; puts a;",
"a = \"hello from a variable\"; puts a;"
```

6.2.14 Screen Output Test

```
"puts \"hello world\";",
"puts 23;",
'puts "hello, " + "world!";',
'puts 3 + 5;'
```

6.2.15 Graph Tests

```
'data = [[1, 3], [3, 4], [5, 5], [10, 6], [12, 10]];
graph({"title" => "Some Random Points",
"type" => "createScatterPlot",
"xLabel" => "the x axis",
"yLabel" => "the y axis",
"legendLabel" => "random stuff",
"data" => data,
"file" => "test_scatter.jpg",
"legend" => true,
"width" => 450,
"height" => 450
});   '
```

6.2.16 Chart Tests

```
'data = {"Graphr" => 10,
"ruby" => 8,
"python" => 5,
"java" => 1};
puts data;
args = {"title" => "Programming languages",
"type" => "createPieChart",     "xLabel" => "ignored",
```

```
"yLabel" => "ignored",
"data" => data,
"file" => "test_pie.jpg",
"legend" => true,
"width" => 450,
"height" => 450
};
puts args;
chart(args);'
```

6.2.17 Include Tests

```
"include
\"test_include_variables.gr\";
puts a;",
"include \"test_include_execute.gr\";
puts \"done with the include\";",
"include \"test_include_function.gr\";
included_function(\"called an included function\");"
```

6.3 Selection of Test Cases

The test cases were chosen so that each case would isolate one particular feature which our language promises and either succeed or fail based solely on whether or not that feature actually functions the way we predict it should. We created test cases which are meant to reflect what a user is likely to do, and test cases which are unlikely to ever be implemented, but must technically work according to our language reference manual. We fixed all bugs in a timely manner, prioritizing the cases we thought were common ahead of those which we thought were unusual.

6.4 Automation

We wrote a Ruby script which could, on command, execute all test cases or certain select test cases.

6.5 Who Did What

Joseph Kamien tested the parser, lexer, and tree-walker concurrent with their development. Paul Dix created the automated Ruby script to assist in testing.

Chapter 7

Lessons Learned

In this chapter, each group member shares his lessons learned and gives advice for future teams.

7.1 Joseph Kamien

Start early. Prioritize the features you want to add, completing essential features first and then adding additional ones later. When deciding which features to promise, maintain focus on the objective of your project, and do not try to include features merely because they seem "cool." Do not make assumptions about what the other members of the team are working on; communicate with them frequently. Examine the code of your fellow group members. If you do not understand something another group member does with his or her code, ask about it. There is a chance your team member solved a problem in a really intelligent way which would not have occurred to you, in which case you might learn something. In the other case, your team member did something nonsensical, and this is a chance for you to help. When doing a bug test, make sure you do a version control update beforehand to make sure that you are not finding bugs that have already been fixed. When you have fixed a bug, commit the code as soon as possible so that other group members are not frustrated by the same bug that you have already eliminated. Commit your code to version control before leaving a room, so that if a laptop is somehow destroyed by a colleague with imprecise knowledge of fluid dynamics, your code is still preserved.

7.2 Paul Dix

Start earlier. Output and plan the AST you'll produce as you write the lexer and parser grammar. We should have re-evaluated the graphics package thing much sooner. Basically we should have planned to revisit each portion of the language as we progressed. You always plan to do much more than you can

actually get done (I knew this, but every major project is just a reminder). Writing a dynamically typed language is hard.

7.3 Zhe Chen

Getting good communication between team members is essential. I think in general our group was pretty good with telling each other what's up with the current status, and even if things didn't work out, as long as everyone was aware certain parts needed work, things could be fixed fairly quickly. Also, it's also crucial to not over state your goals in the beginning, as we were a bit too ambitious in what we planned to do, and in the end, although we got the major functionality done, we were not able to implement a lot of the smaller things we said we'd do. However, because we left our code flexible, a lot of those things could still be easily implemented if more time is spent, or the user himself should be able to write them using what we provided. So definitely, it's good that we made our language flexible enough so that even though it only has the basic functionality right now, the user can use it to make much more complex programs that do what they want it to do. Also, it's a good idea to look ahead a bit more. All the code I wrote for the drawing portion of the language ended up not making it in, as wiring it up would have taken a lot of time, and we still had to write the libraries that used those drawing tools to create graphs. We ended using an open source graph drawing library to replace it, as it was much easier to wire up and had all the functionality already. Had we decided to go with the open source graph drawing library first, we could have saved a lot of time from designing our own graph drawing tools and spent more time implementing the extra functionality we had to cut.

7.4 Michael Cole

Like every PLT group before us, we probably should have started earlier. When writing our LRM, we did not spend nearly enough time working out our lexer and parser issues. This error in judgment pushed everything else for the project back by at least two weeks. I have also learned that we should have aimed even lower than we originally did. We definitely bit off more than we could chew despite the fact that we were trying to weary of adding extraneous functionality. I cannot help but feel slightly disappointed that we did not meet all of our claims in our LRM, despite the fact that we implemented the majority of what we aspired for in the LRM.

Appendix A

Complete Log

A.1 Subversion Log

Revision: 110
Author: whatisthelemon
Date: 4:29:00 PM, Tuesday, December 18, 2007
Message:
Changed the testing section.
Note that this version describes our language as dynamically scoped. If we change that, it's ok to revert back to this version of the final doc.
—-
Modified : /trunk/Docs/final.lyx
Revision: 109
Author: whatisthelemon
Date: 10:32:39 PM, Monday, December 17, 2007
Message:
Added information about our testing
—-
Modified : /trunk/Docs/final.lyx
Revision: 108
Author: whatisthelemon
Date: 9:31:45 PM, Monday, December 17, 2007
Message:
Made changes to architectural design section.
—-
Deleted : /trunk/Docs/block_compiler.JPG
Added : /trunk/Docs/block_compiler3.jpg
Modified : /trunk/Docs/final.lyx
Revision: 107
Author: whatisthelemon
Date: 1:55:50 PM, Monday, December 17, 2007

Message:
Adding data file
—-
Added : /trunk/antlr/data.csv
Revision: 106
Author: whatisthelemon
Date: 1:53:03 PM, Monday, December 17, 2007
Message:
Added file manip stuff to the demo
—-
Modified : /trunk/antlr/demo.gr
Modified : /trunk/antlr/graphlib.gr
Revision: 105
Author: paulcdix
Date: 1:48:33 PM, Monday, December 17, 2007
Message:
* fixed creating of an array with no elements
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 104
Author: whatisthelemon
Date: 1:42:48 PM, Monday, December 17, 2007
Message:
Added files for demo.
—-
Added : /trunk/antlr/demo.gr
Added : /trunk/antlr/graphlib.gr
Revision: 103
Author: paulcdix
Date: 1:33:44 PM, Monday, December 17, 2007
Message:
* put addition into GraphrArray
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrArray.java
Modified : /trunk/antlr/all_tests.rb
Revision: 102
Author: xionvalkyrie
Date: 12:22:09 AM, Monday, December 17, 2007
Message:
—-
Modified : /trunk/Docs/GraphrTut.pdf
Revision: 101

Author: xionvalkyrie
Date: 12:16:45 AM, Monday, December 17, 2007
Message:
Added Slides for tutorial
—-
Added : /trunk/Docs/GraphrTut.pdf
Revision: 100
Author: whatisthelemon
Date: 11:37:55 PM, Sunday, December 16, 2007
Message:
Edited and added Zhe's stuff (changes to the LRM to reflect our graphics library).
—-
Modified : /trunk/Docs/final.lyx
Deleted : /trunk/Docs/zhelessonslearned1.lyx
Revision: 99
Author: xionvalkyrie
Date: 11:29:33 PM, Sunday, December 16, 2007
Message:
Added graphics Library to the LRM
—-
Modified : /trunk/Docs/final.lyx
Revision: 98
Author: whatisthelemon
Date: 10:39:25 PM, Sunday, December 16, 2007
Message:
Added lessons learned to the final report, and added the full svn log.
—-
Added : /trunk/Docs/block_compiler.JPG
Modified : /trunk/Docs/final.lyx
Added : /trunk/Docs/svn_log.txt
Added : /trunk/Docs/test_pie.jpg
Added : /trunk/Docs/test_scatter.jpg
Revision: 97
Author: xionvalkyrie
Date: 9:39:27 PM, Sunday, December 16, 2007
Message:
—-
Modified : /trunk/antlr/DrawCanvas.java
Modified : /trunk/antlr/writeImage.java
Revision: 96
Author: xionvalkyrie
Date: 9:35:08 PM, Sunday, December 16, 2007
Message:
—-
Added : /trunk/Docs/zhelessonslearned1.lyx

Revision: 95
Author: whatisthelemon
Date: 9:31:33 PM, Sunday, December 16, 2007
Message:
—-
Modified : /trunk/Docs/final.lyx
Revision: 94
Author: xionvalkyrie
Date: 7:04:12 PM, Sunday, December 16, 2007
Message:
—-
Added : /trunk/antlr/test_pie.jpg
Added : /trunk/antlr/test_scatter.jpg
Revision: 93
Author: whatisthelemon
Date: 6:50:07 PM, Sunday, December 16, 2007
Message:
Spellchecked and edited.
—-
Modified : /trunk/Docs/tut.lyx
Revision: 92
Author: whatisthelemon
Date: 6:49:39 PM, Sunday, December 16, 2007
Message:
Spellchecked and edited Zhe's tutorial part, then added it to document.
—-
Modified : /trunk/Docs/final.lyx
Revision: 91
Author: whatisthelemon
Date: 6:27:52 PM, Sunday, December 16, 2007
Message:
Made changes to include more of our coding conventions.
—-
Modified : /trunk/Docs/final.lyx
Revision: 90
Author: xionvalkyrie
Date: 6:26:39 PM, Sunday, December 16, 2007
Message:
Added the tutorial section for the final paper.
—-
Added : /trunk/Docs/tut.lyx
Revision: 89
Author: whatisthelemon
Date: 4:42:07 PM, Sunday, December 16, 2007
Message:
Did some fixes to LRM

—-
Modified : /trunk/Docs/final.lyx
Revision: 88
Author: paulcdix
Date: 4:39:59 PM, Sunday, December 16, 2007
Message:
* fixed a bug with foreach statements that would cause the program to end after they're done
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 84
Author: paulcdix
Date: 3:58:55 PM, Sunday, December 16, 2007
Message:
* fixed a bug that was introduced earlier that broke hashes
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Revision: 83
Author: paulcdix
Date: 3:57:34 PM, Sunday, December 16, 2007
Message:
* fixed the logical stmt so you can do if (t == 1)
* this removed the ability to do if (3) ... it's stupid, don't do it
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 82
Author: paulcdix
Date: 3:45:40 PM, Sunday, December 16, 2007
Message:
* wired up the == operator
* wired up the != operator
* added a bunch of equals code to the graphr data types
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrArray.java
Modified : /trunk/antlr/GraphrAssociativeArray.java
Modified : /trunk/antlr/GraphrDataType.java
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb

Revision: 80
Author: paulcdix
Date: 3:05:38 PM, Sunday, December 16, 2007
Message:
* wired up the not ! operator
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrBoolean.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 79
Author: paulcdix
Date: 2:48:47 PM, Sunday, December 16, 2007
Message:
* added the mod operation to GraphrString
* fixed mod in GraphrNumber to try to convert the rval to a number if it's a string and removed a dubug line
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Modified : /trunk/antlr/all_tests.rb
Revision: 78
Author: paulcdix
Date: 2:42:48 PM, Sunday, December 16, 2007
Message:
* wired subtract, multiply, divide and mod to strings and numbers
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrDataType.java
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 76
Author: paulcdix
Date: 2:16:31 PM, Sunday, December 16, 2007
Message:
* added another parenthesis option to logical and add stmts
* fixed while and if to look for logical_stmts
* sacrificed a kitten to the antlr gods
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrWalker.g

Revision: 71
Author: paulcdix
Date: 2:00:05 PM, Sunday, December 16, 2007
Message:
* wired up return statements in functions
* fixed parsing rules for logical_stmt
* put func_call_stmt into enclosed_stmt to fix add_stmt precendence
* put logical_stmt into enclosed_stmt to fix add_stmt precendence
* made all values that to something other than null or false return true
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 68
Author: blakstar1220
Date: 7:06:10 PM, Saturday, December 15, 2007
Message:
Documentation added. Except for WriteImage and DrawCanvas.
—-
Modified : /trunk/antlr/Graphr.java
Modified : /trunk/antlr/GraphrArray.java
Modified : /trunk/antlr/GraphrAssociativeArray.java
Modified : /trunk/antlr/GraphrBoolean.java
Modified : /trunk/antlr/GraphrCollection.java
Modified : /trunk/antlr/GraphrDataType.java
Modified : /trunk/antlr/GraphrGraphics.java
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Revision: 67
Author: paulcdix
Date: 4:29:27 PM, Saturday, December 15, 2007
Message:
* wired up string functions split, length, and substring
* added tests for the new functions
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 66
Author: paulcdix
Date: 6:05:06 PM, Thursday, December 13, 2007
Message:
* wired up the graphics library
* fixed a bug in the graphics library related to some arg named Locale

* added a test for chart and graph
*
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrAssociativeArray.java
Modified : /trunk/antlr/GraphrGraphics.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/Manifest.txt
Modified : /trunk/antlr/all_tests.rb
Modified : /trunk/antlr/build_and_test.rb
Revision: 65
Author: blakstar1220
Date: 4:58:07 PM, Thursday, December 13, 2007
Message:
JAR's for JFreeChart
—-
Added : /trunk/antlr/gnujaxp.jar
Added : /trunk/antlr/itext-2.0.6.jar
Added : /trunk/antlr/jcommon-1.0.12.jar
Added : /trunk/antlr/jfreechart-1.0.8a-experimental.jar
Added : /trunk/antlr/jfreechart-1.0.8a-swt.jar
Added : /trunk/antlr/jfreechart-1.0.8a.jar
Added : /trunk/antlr/junit.jar
Added : /trunk/antlr/servlet.jar
Revision: 64
Author: blakstar1220
Date: 4:56:01 PM, Thursday, December 13, 2007
Message:
Charts and graphs added.
—-
Added : /trunk/antlr/GraphrGraphics.java
Revision: 63
Author: paulcdix
Date: 4:30:38 PM, Thursday, December 13, 2007
Message:
* added code to handle addition of numbers and strings better
* made GraphrNumber output a better string on toString()
—-
Modified : /trunk/antlr/Graphr.jar
Modified : /trunk/antlr/GraphrDataType.java
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Revision: 62
Author: paulcdix
Date: 3:48:22 PM, Thursday, December 13, 2007

Message:
* added the hello world script
* added the Mainifest for building a single jar
* added Graphr.java which is the main interpreter class
* removed debugging lines from the Walker
* modified build script to package everything up into a single jar
* and added Graphr.jar which is the binary of the full interpreter
—-
Added : /trunk/antlr/Graphr.jar
Added : /trunk/antlr/Graphr.java
Modified : /trunk/antlr/GraphrWalker.g
Added : /trunk/antlr/Manifest.txt
Modified : /trunk/antlr/build_and_test.rb
Added : /trunk/antlr/hello_world.gr
Revision: 61
Author: paulcdix
Date: 3:00:39 PM, Thursday, December 13, 2007
Message:
* wired up include into the parser and walker
* added tests for include
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Added : /trunk/antlr/test_include_execute.gr
Added : /trunk/antlr/test_include_function.gr
Added : /trunk/antlr/test_include_variables.gr
Revision: 60
Author: xionvalkyrie
Date: 5:30:47 AM, Thursday, December 13, 2007
Message:
Added some comments to the code, also added WriteImageType.java as an example of how to write an image once you have a hashmap.
—-
Added : /trunk/antlr/WriteImageType.java
Modified : /trunk/antlr/writeImage.java
Revision: 53
Author: whatisthelemon
Date: 2:26:34 AM, Wednesday, December 12, 2007
Message:
Made changes which do not affect program, but ensure that we are following our own coding conventions.
—-
Modified : /trunk/antlr/all_tests.rb
Revision: 52
Author: whatisthelemon

Date: 3:51:36 PM, Tuesday, December 11, 2007
Message:
Started to work on our final document. Needs sections on Architectural Design, test plan, Lessons Learned, Appendix, and Language Tutorial. Also, our White Paper and Language Manual need to be updated and finalized.
—-
Added : /trunk/Docs/final.lyx
Revision: 51
Author: paulcdix
Date: 3:50:15 PM, Tuesday, December 11, 2007
Message:
* added tests for function definitions and calls
* wired up function calls in the parser and walker
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/TestGraphr.java
Modified : /trunk/antlr/all_tests.rb
Revision: 50
Author: xionvalkyrie
Date: 3:19:23 PM, Tuesday, December 11, 2007
Message:
—-
Deleted : /trunk/graphics
Revision: 49
Author: xionvalkyrie
Date: 3:19:07 PM, Tuesday, December 11, 2007
Message:
—-
Deleted : /trunk/antlr/WriteImageType.java
Revision: 48
Author: xionvalkyrie
Date: 3:18:42 PM, Tuesday, December 11, 2007
Message:
Added vertical text functionality
—-
Modified : /trunk/antlr/DrawCanvas.java
Added : /trunk/antlr/WriteImageType.java
Modified : /trunk/antlr/writeImage.java
Revision: 47
Author: xionvalkyrie
Date: 2:20:31 PM, Tuesday, December 11, 2007
Message:
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 46

Author: paulcdix
Date: 2:13:02 PM, Tuesday, December 11, 2007
Message:
* added tests for file io
* added parsing rules for file io
* added functionality to tree grammar for file io
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Added : /trunk/antlr/test.txt
Revision: 45
Author: xionvalkyrie
Date: 1:29:17 PM, Tuesday, December 11, 2007
Message:
Updated the LRM to reflect changes in the graphics engine.
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 44
Author: xionvalkyrie
Date: 1:21:25 PM, Tuesday, December 11, 2007
Message:
Completed the interfacing of Graphr Associative Array to the graphics functions, so now we can take a Graphr Associative Array as input and output the corresponding image file.
—-
Modified : /trunk/antlr/writeImage.java
Revision: 43
Author: paulcdix
Date: 12:49:53 PM, Tuesday, December 11, 2007
Message:
* wired up hash and array access
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 42
Author: xionvalkyrie
Date: 12:18:10 AM, Tuesday, December 11, 2007
Message:
—-
Added : /trunk/antlr/DrawCanvas.java
Added : /trunk/antlr/writeImage.java
Revision: 41
Author: xionvalkyrie
Date: 12:17:17 AM, Tuesday, December 11, 2007

Message:
—-
Modified : /trunk/graphics/src/WriteImageType.java
Modified : /trunk/graphics/src/writeImage.java
Revision: 40
Author: paulcdix
Date: 5:49:52 PM, Sunday, December 09, 2007
Message:
* added the collection interface
* modified hashes so they only take strings as keys
* modified the GraphrAssociativeArray to only accept strings as keys
* did some other stuff
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrArray.java
Modified : /trunk/antlr/GraphrAssociativeArray.java
Added : /trunk/antlr/GraphrCollection.java
Modified : /trunk/antlr/GraphrDataType.java
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 39
Author: paulcdix
Date: 3:57:38 PM, Sunday, December 09, 2007
Message:
* added the GraphrBoolean class
* added some tests for hashes
* fixed the math operations
* fleshed out methods on the GraphrAssociativeArray class
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrAssociativeArray.java
Added : /trunk/antlr/GraphrBoolean.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Modified : /trunk/antlr/build_and_test.rb
Revision: 38
Author: xionvalkyrie
Date: 2:54:02 AM, Monday, December 03, 2007
Message:
—-
Modified : /trunk/graphics/src/DrawCanvas.java
Modified : /trunk/graphics/src/WriteImageType.java
Added : /trunk/graphics/src/writeImage.java
Revision: 37

Author: paulcdix
Date: 5:33:16 PM, Saturday, December 01, 2007
Message:
* fixed the toString method on GraphrArray
* fixed some grammar errors
* added tests for loops and if statements
* wired up loops and if statements
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrArray.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 36
Author: paulcdix
Date: 11:49:43 AM, Monday, November 26, 2007
Message:
* updated the tree walker
* updated the Graphr data types
* added some tests
—-
Modified : /trunk/antlr/Graphr.g
Modified : /trunk/antlr/GraphrArray.java
Modified : /trunk/antlr/GraphrDataType.java
Modified : /trunk/antlr/GraphrNumber.java
Modified : /trunk/antlr/GraphrString.java
Modified : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/all_tests.rb
Revision: 35
Author: xionvalkyrie
Date: 3:23:14 PM, Sunday, November 25, 2007
Message:
—-
Modified : /trunk/graphics/src/DrawCanvas.java
Modified : /trunk/graphics/src/WriteImageType.java
Revision: 34
Author: xionvalkyrie
Date: 2:34:27 PM, Sunday, November 25, 2007
Message:
—-
Added : /trunk/graphics/src/DrawCanvas.java
Modified : /trunk/graphics/src/WriteImageType.java
Revision: 33
Author: paulcdix
Date: 1:50:19 PM, Sunday, November 25, 2007
Message:
* initial add of GraphrWalker and Graphr java data types

* made some changes to the grammar to work for walker
—-
Modified : /trunk/antlr/Graphr.g
Added : /trunk/antlr/GraphrArray.java
Added : /trunk/antlr/GraphrAssociativeArray.java
Added : /trunk/antlr/GraphrDataType.java
Added : /trunk/antlr/GraphrNumber.java
Added : /trunk/antlr/GraphrString.java
Added : /trunk/antlr/GraphrWalker.g
Modified : /trunk/antlr/TestGraphr.java
Modified : /trunk/antlr/all_tests.rb
Modified : /trunk/antlr/build_and_test.rb
Revision: 32
Author: paulcdix
Date: 12:10:38 AM, Monday, November 19, 2007
Message:
* added the build and test script
* added the java test harness
* rewrote most of Graphr.g to generate a workable AST
* added the jar files needed for build scripts
* added all_tests.rb (a ruby script containing some tests)
—-
Modified : /trunk/antlr/Graphr.g
Added : /trunk/antlr/TestGraphr.java
Added : /trunk/antlr/all_tests.rb
Added : /trunk/antlr/antlr-2.7.7.jar
Added : /trunk/antlr/antlr-3.0.1.jar
Added : /trunk/antlr/build_and_test.rb
Added : /trunk/antlr/stringtemplate.jar
Revision: 31
Author: xionvalkyrie
Date: 12:44:18 AM, Monday, November 05, 2007
Message:
11-04-07
Added graphics
—-
Added : /trunk/graphics
Added : /trunk/graphics/.classpath
Added : /trunk/graphics/.project
Added : /trunk/graphics/bin
Added : /trunk/graphics/bin/WriteImageType.class
Added : /trunk/graphics/src
Added : /trunk/graphics/src/WriteImageType.java
Added : /trunk/graphics/test.PNG
Revision: 30
Author: blakstar1220

Date: 3:53:37 PM, Wednesday, October 24, 2007
Message:
Updated Grammar. All issues handled except for exponents...
—-
Modified : /trunk/antlr/Graphr.g
Revision: 29
Author: blakstar1220
Date: 11:08:20 PM, Tuesday, October 23, 2007
Message:
Almost completed parser. Things to fix/add:
1. Functionality of testLiterals for identifiers.
2. The dangling else problem
3. Exponentiation rule not implemented.
4. General structure of the tree. (Paul must decide this)
—-
Modified : /trunk/antlr/Graphr.g
Revision: 28
Author: blakstar1220
Date: 1:26:05 PM, Monday, October 22, 2007
Message:
First iteration of the scanner. Problems with assignment and if statements.
—-
Modified : /trunk/antlr/Graphr.g
Revision: 27
Author: paulcdix
Date: 9:32:55 PM, Wednesday, October 17, 2007
Message:
* changed IO function names a little bit
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 26
Author: paulcdix
Date: 9:28:36 PM, Wednesday, October 17, 2007
Message:
* added the section on the graphics
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 25
Author: xionvalkyrie
Date: 8:33:07 PM, Wednesday, October 17, 2007
Message:
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 24
Author: xionvalkyrie
Date: 8:30:12 PM, Wednesday, October 17, 2007

Message:
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 23
Author: xionvalkyrie
Date: 8:23:10 PM, Wednesday, October 17, 2007
Message:
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 22
Author: paulcdix
Date: 7:53:08 PM, Wednesday, October 17, 2007
Message:
* added information on variable scope
* added info on function scope
* added foreach
* started library section
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 21
Author: whatisthelemon
Date: 5:38:14 PM, Tuesday, October 16, 2007
Message:
Since we aren't sure if we will include regular expressions, I moved the regexp stuff into a different file.
—-
Modified : /trunk/Docs/lrm.lyx
Added : /trunk/Docs/regexp.lyx
Revision: 20
Author: xionvalkyrie
Date: 5:10:17 PM, Monday, October 15, 2007
Message:
—-
Modified : /trunk/antlr/Graphr.g
Revision: 19
Author: whatisthelemon
Date: 4:59:04 PM, Monday, October 15, 2007
Message:
Changed the formatting. Fixed some typos. Used quotation environment whenever describing example syntax. More stuff needs to be done for regular expressions (we need to explain order of precedence).
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 18
Author: whatisthelemon
Date: 3:58:26 PM, Monday, October 15, 2007

Message:
Updated up to regular expressions.
—-
Modified : /trunk/Docs/lrm.lyx
Revision: 17
Author: xionvalkyrie
Date: 3:54:45 PM, Monday, October 15, 2007
Message:
—-
Modified : /trunk/antlr/Graphr.g
Revision: 16
Author: xionvalkyrie
Date: 2:16:20 PM, Monday, October 15, 2007
Message:
Fixed assignment_expr
Left side must be an identifier_expr
No longer recursive to itself
—-
Modified : /trunk/antlr/Graphr.g
Revision: 15
Author: paulcdix
Date: 2:16:08 PM, Monday, October 15, 2007
Message:
* added the lyx file and modified the tex file a little.
—-
Added : /trunk/Docs/lrm.lyx
Modified : /trunk/Docs/lrm.tex
Revision: 14
Author: whatisthelemon
Date: 4:15:28 PM, Sunday, October 14, 2007
Message:
Added more. Still have a lot more to do. We have to decide how exactly we're dealing with functions. The old lrm talked a lot about graph objects, but a graph is something final that's printed to the user in a way that may or may not be interactive. We should think more in terms of objects for representing data internally, like rows, columns, and tables.
—-
Modified : /trunk/Docs/lrm.tex
Revision: 13
Author: whatisthelemon
Date: 3:42:12 PM, Sunday, October 14, 2007
Message:
Started porting to latex, still have a long way to go.
—-
Added : /trunk/Docs/lrm.tex
Revision: 12

Author: blakstar1220
Date: 6:34:17 PM, Saturday, October 13, 2007
Message:
Updated G file. Full of bugs.
—-
Modified : /trunk/antlr/Graphr.g
Revision: 11
Author: whatisthelemon
Date: 6:33:03 PM, Saturday, October 13, 2007
Message:
Made changes regarding expressions. We tried to get rid of the hard ones.
—-
Modified : /trunk/Docs/ReferenceManual.rtf
Revision: 10
Author: whatisthelemon
Date: 4:06:40 PM, Saturday, October 13, 2007
Message:
Edited a bunch of stuff since statements are evaluated, not just expressions.
—-
Modified : /trunk/Docs/ReferenceManual.rtf
Revision: 9
Author: paulcdix
Date: 3:54:54 PM, Saturday, October 13, 2007
Message:
* added the initial version of the graphr gramamar!
—-
Added : /trunk/antlr
Added : /trunk/antlr/Graphr.g
Revision: 8
Author: xionvalkyrie
Date: 2:11:18 PM, Saturday, October 13, 2007
Message:
—-
Modified : /trunk/Docs/ReferenceManual.rtf
Revision: 7
Author: whatisthelemon
Date: 2:04:42 PM, Saturday, October 13, 2007
Message:
Removed part about keeping newlines in whitespace.
—-
Modified : /trunk/Docs/ReferenceManual.rtf
Revision: 6
Author: paulcdix
Date: 2:02:31 PM, Saturday, October 13, 2007
Message:
* removed the odt and pdfs of the ref manual

—-
Deleted : /trunk/Docs/ReferenceManual.odt
Deleted : /trunk/Docs/ReferenceManual.pdf
Revision: 5
Author: xionvalkyrie
Date: 1:59:06 PM, Saturday, October 13, 2007
Message:
—-
Added : /trunk/Docs/ReferenceManual.rtf
Revision: 4
Author: xionvalkyrie
Date: 1:57:55 PM, Saturday, October 13, 2007
Message:
—-
Added : /trunk/Docs/ReferenceManual.odt
Revision: 3
Author: xionvalkyrie
Date: 1:57:07 PM, Saturday, October 13, 2007
Message:
—-
Added : /trunk/Docs/ReferenceManual.pdf
Revision: 2
Author: xionvalkyrie
Date: 1:55:18 PM, Saturday, October 13, 2007
Message:
—-
Added : /trunk/Docs
Revision: 1
Author:
Date: 12:21:17 PM, Saturday, October 13, 2007
Message:
Initial directory structure.
—-
Added : /branches
Added : /tags
Added : /trunk

A.2 Google Code Log

Issue 1: [Vote for this issue and get email change notifications] read_file return 1 of 4 Next › 1 person starred this issue and may be notified of changes. Back to list Status: Fixed Owner: whatisthelemon Closed: Today Type-Defect Priority-High

Add a comment and make changes below Reported by whatisthelemon, Today (4 hours ago)

We say that read_file() will return nil if the file does note exist. Hence the following code:

x = read_file("input.txt") // input.txt does not exist if(x){ puts "Read file successful"; }

should output nothing, but it outputs "Read file successful"

Delete comment Comment 1 by whatisthelemon, Today (4 hours ago)

This has been changed so that it read_file() will return an empty array if the file does not exist.

Delete comment Comment 2 by whatisthelemon, Today (3 hours ago)

(No comment was entered for this change.)

Status: Fixed Delete comment Comment 3 by whatisthelemon, Today (3 hours ago)

(No comment was entered for this change.)

Status: Started Delete comment Comment 4 by whatisthelemon, Today (2 hours ago)

(No comment was entered for this change.)

Status: Fixed

Issue 2: [Vote for this issue and get email change notifications] Comparison Expressions ‹ Prev 2 of 4 Next › 1 person starred this issue and may be notified of changes. Back to list Status: Verified Owner: whatisthelemon Closed: Today Type-Defect Priority-High

Add a comment and make changes below Reported by whatisthelemon, Today (4 hours ago)

Does not seem to properly evaluate comparisons.

The following code:

t = 1; if(t == 1){ puts t; } else{ puts 0; }

should print "1" but instead it prints "0".

Same problem occurs if I try $t > 0$ or $t < 5$ (it should print 1 but it prints 0).

Delete comment Comment 1 by whatisthelemon, Today (3 hours ago)

It's been resolved for all such comparisons.

Issue 3: [Vote for this issue and get email change notifications] Return does not end function call ‹ Prev 3 of 4 Next › 1 person starred this issue and may be notified of changes. Back to list Status: WontFix Owner: whatisthelemon Closed: Today Type-Defect Priority-High

Add a comment and make changes below Reported by whatisthelemon, Today (4 hours ago)

The following code:

def myfun(){ t = true; if(t){ return 1; } puts "I have not returned"; }

x = myfun(); puts x;

Should print 1 and nothing else. Instead, it prints: I have not returned 1

Getting to a return statement seems to return the proper value, but does not end the function call.

This can cause a serious problem because it can result in more than one return statement being called.

The following code causes an EmptyStackException:

def myfun(){ t = true; if(t){ return 1; } return 2; }
x = myfun(); puts x;
The following code works as expected:
def myfun(){ t = true; if(t){ return 1; } else{ puts "I have not returned"; } }
x = myfun(); puts x;
Delete comment Comment 1 by whatisthelemon, Today (3 hours ago)
No function should have more than one return statement (it is a bad programming practice according to my 1007 teacher Prof. Cannon). We're making it a requirement that each function can have only one return statement.
Status: WontFix
Issue 4: [Vote for this issue and get email change notifications] foreach loop terminates program ‹ Prev 4 of 4 1 person starred this issue and may be notified of changes. Back to list Status: Fixed Owner: whatisthelemon Closed: Today Type-Defect Priority-Medium
Add a comment and make changes below Reported by whatisthelemon, Today (3 hours ago)
When reaching the end of a foreach loop, the program terminates. Other loops appear to be working fine.
Delete comment Comment 1 by whatisthelemon, Today (2 hours ago)
Small typo in code. Works now.
Status: Fixed

Appendix B

Code Listing

This is a complete listing of code which the team generated, organized by person. We do not include code generated automatically by ANTLR, only our original .g files. All files listed are available on our our Subversion repository, which can be accessed at `http://graphr.googlecode.com/svn/trunk/antlr/`.

B.1 Joseph Kamien

As the tester and documenter, Joseph Kamien has no original files to call his own other than the documentation (including this report) and the aforementioned test cases and programs. He worked with the other team members to constantly attempt to break their code as it was being developed.

B.2 Michael Cole

Michael Cole was the main creator of our lexer and parser.
He is the main author of the following files:

Graphr.g

GraphrGraphics.java

B.3 Paul Dix

Paul Dix was the main creator of our tree-walker.
He is the main author of the following files:

GraphrWalker.g

build_and_test.rb

all_tests.rb

GraphrDataType.java

GraphrNumber.java

GraphrString.java

GraphrCollection.java

GraphrArray.java

GraphrAssociativeArray.java

Graphr.java

B.4 Zhe Chen

Zhe Chen was the main creator of our graphics package.
He is the main author of the following files:

DrawCanvas.java

WriteImage.java