GraphQuil

Final Report
COMS 4115 Programming Languages and Translators
Professor Stephen Edwards

John Heizelman (jrh2184)
Jon Paul (jp3144)
GemmaRose Ragozzine (gr2390)
Steven Weiner (srw2163)

Table Of Contents

1 Introduction

GraphQuil is a specialized programming language designed to easily manage and manipulate data stored in graphs. It is simultaneously a data-definition language as well as a data manipulation language. Users operating GraphQuil will have the ability to create vertices that represent objects while also being able to link them together and define their relationships within a graph.

2 Language Tutorial

2.1 Using the compiler

2.1.1 Installing the GraphQuil compiler

To install the GraphQuil compiler, the git version control tool is required.

2.1.2 Running the GraphQuil compiler

Once downloaded, the compiler takes one of five flags upon start up. The five valid flags are as follows:

-a	Print AST of source
-t	Print Symbol Table of source
-s	Run Semantic Analysis over source
-i	Print Intermediate Representation of source
-c	Compile source

Here is an example of using the five command options with the sample program, helloWorld.gq. This sample program is included in section 6.1 (Sample Programs).

- Print AST of helloWorld.gq
  ```
  ./graphquil –a tests/helloWorld.gq
  ```
- Print Symbol Table of helloWorld.gq
  ```
  ./graphquil –t tests/helloWorld.gq
  ```
- Run Semantic Analysis over helloWorld.gq
  ```
  ./graphquil –s tests/helloWorld.gq
  ```
- Print Intermediate Representation of helloWorld.gq

```
./graphquil –i tests/helloWorld.gq
```

- Compile helloWorld.gq
  ```
  ./graphquil –c tests/helloWorld.gq
  ```

2.2 Program structure

Each GraphQuil program must contain a main() function that does not return anything. For clarity, the main function cannot even have a return statement within it. The signature of the main() function is as follows:

```
void main(){}
```

Note that the signature of main is also a valid declaration of main, since the function never includes a return statement.

The entire GraphQuil program must be contained in one *.gq file.

3 Language Reference Manual

3.1 Types

3.1.1 Explicit Typing

Data in GraphQuil requires explicit typing. In naming any variable type, an explicit specification of type is required.

3.1.2 Primitive Types

GraphQuil provides four primitive types that make up the basis for arranging data in your programs: *int*, *char*, *String*, and *bool*. These are the building blocks that define what type of data can be included in each Node.

3.1.2.1 Ints

Integers, denoted by the keyword *int* are 32-bit numbers with a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).

3.1.2.2 Chars

Chars, denoted by the keyword *char*, is an 8-bit data type used specifically for storing ASCII characters.

3.1.2.3 Booleans

Booleans, denoted by the keyword *bool*, accept only *true* or *false* values and are used in comparisons of equality.

3.1.3 Non-Primitive Types

3.1.3.1 Strings

Strings are implemented as arrays of *chars*. Whitespace is not ignored. Here is an example of the initializing of a String called "string":

```
String stringExample = "string";
```

The string stores six *chars*, and so its size is at least 48 bits.

3.1.3.2 Nodes

A *Node* is a data structure specifically for use in a *Graph*. Nodes are required to store one array, *edges*, that stores all the outward edges from the node. This array is special in that it may not be modified directly; it is modified automatically on the linking of two nodes, as seen in section 4.2.6. Additionally, nodes can only be present in one graph at a time, and a duplicate node with the same data must be created before adding it to a different graph. To meet this requirement, the graph must be specified when creating the node.

Each node must also be given its own type before creation, and there is no base implementation of Node within GraphQuil that stores certain variables. Before declaring a node, there must be a defined NodeType to denote what that specific kind of node will hold (see section 3.4 for more information on NodeTypes). However, links can be made irrespective of type, and the type of node does not affect the way links are created between them in the graph.

3.1.3.3 Edges

An edge is a link created between two nodes. There are two different kinds of edge types, weighted and unweighted. Unweighted edges create links that connect two nodes, but provide no information about those links. Weighted edges, however, in a way similar to nodes, can contain all kinds of data about the connections between different nodes. To create a weighted edge between two nodes, there must first be a defined EdgeType (see section 3.5 for more info), then a declaration of the edge that specifies what type of edge to be created and the data to populate said edge (see section 4.2.7 for the syntax for doing so).

3.2 - Lexical Conventions

3.2.1 Identifiers (IDs)

Identifiers are the names used to identify individual variables within a given function. Each variable must be given a unique identifier within its scope of the function, and each identifier is case-sensitive. These identifiers must begin with an ascii letter (a-z or A-Z), but each following letter can be either a letter, a digit (0-9), or an underscore (_). The only additional restriction on what strings can be used as identifiers is that they must not match one of the GraphQuil's reserved keywords.

3.2.2 Reserved Keywords

The following is a comprehensive list of reserved keywords inGraphQuil:

add	bool	char
continue	dest	do
double	edges	else
false	for	fal
if	in	while
new	Node	int
print	return	String
true	void	false

3.2.3 Literals

3.2.3.1 Integer Literals

Integer literals are made up of digits without any decimal points or other characters separating the digits. They are solely to be used for whole numbers, but they can be positive or negative, distinguished by an included or omitted '-' sign. Additionally, integer literals may not begin with a 0.

3.2.3.2 Bool Literals

The bool type can hold two different values, *true* or *false*, and those exact literals are used to declare which value the bool takes.

3.3.4 Char Literals

Char literals are a single character surrounded by a single quote (') on each side. This character can be any character in the 16-bit Unicode character set, with a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

3.3.5 String Literals

String literals are sequences of chars anywhere from length 0 to the maximum integer value (around 2 billion characters). They must begin and end with quotation marks (") and can be filled with any character within the same character set as chars.

3.3 - Node Types

Before declaring a specific node, a node type must be first established, as there is no general type node to be instantiated by the user. To do so, there must be a line defining what type of node the identifier will be associated with. The type includes a listing of all the values to be stored in that node type, as well as the type of values the node stores. These values can be of any valid primitive or non-primitive type, with the exception of graphs, edges, and nodes. Using nodes to store other nodes removes the purpose behind a graph structure, so all data contained within a node must be of some other type.

Additionally, each value must be associated with an identifier to indicate what the name of that value is, and each identifier must be unique for that node type. For example, the following is a valid declaration of a NodeType:

```
#assume there is a defined Graph g1
NodeType Student has (String name, int age, String[] courses);
```

To create a particular instance of a node, the type must be specified in its construction, along with the graph the node is inserted into:

```
        Node john = new Student("John", 21, {"PLT", "CS Theory", "Aristotle"})
in g1;
```

3.4 - Edge Types

Creating a weighted edge requires an edge type to define what kind of data the edge will hold. This structure is very similar to defining a Node, and requires the identifier of the Edge, as well as the types and identifiers for each piece of data to be stored. Also, the same requirement applies in that these values can take any type other than nodes, graphs, or other edges. The convention is as follows:

```
Edge relationship has (String type, int closeness, bool positive);
```

3.5 - Operators

The following denotes the operators of the language and their intended function:

Operator	Use	Associativity
+	Adds two values	Left
-	Substracts right from left	Left
*	Multiplies two values	Left
/	Divides right into left	Left
%	Modular division	Left
=	Assignment	Right
==	Equal to	Left
!=	Not equal to	Left
<	Less than	Left
>	Greater than	Left
<=	Less than or equal to	Left
>=	Greater than or equal to	Left

->	Link from left to right	Right
<->	Bi-directional link	Right
!	Logical negation	Right
&&	Logical and	Left
||	Logical or	Left
add	Adds an attribute to a Node or Edge	Right

The order of precedence is as follows, from greatest to least important:
* / %
+ -
< > <= >=
!
== !=
&&
||
= -> <-> add

3.6 - Separators

The following indicates the separators of the language and their usage.

“;”
Signals the end of a line.
“,”
Separates one argument from another argument in a declaration statement.

“(...)”

A parenthesized expression can be used to separate tokens. The type and value are identical to the same expression without the surrounding parentheses.

3.7 - Whitespace

White space includes the space character, the tab character, and the newline character. Generally used to separate characters, it is entirely optional. No white space is required.

3.8 - Comments

Multi-line comments are enclosed in #* *# characters. The #* characters introduce a multi-line comment and the *# signals the termination of the multi-line comment.

Single line comments are introduced by the character #. The single line comment terminates at the end of the line with no terminating symbol.

4 Syntax

4.1- Program Structure

A valid program in this language can be composed of as little or as many statements that a user desires.

4.2 - Expressions

4.2.1 Unary Operators

!	The character '!' is the logical NOT operator. This operator checks the value of the one boolean operand. If the boolean operand evaluates to true, then the expression returns false. If the boolean operand evaluates to false, the expression returns true.

4.2.1 Binary Operators

The binary operators are divided into give categories: Assignment Operators, Relational Operators, Arithmetic Operators, Logical Operators, and Graphical Operators.

4.2.3 Assignment Operators

All assignment operators group right to left.

=	The simple assignment operator is the '=' character. Simple assignment takes the following form: type-declaration identifier = expression

	The value of the expression on the right hand side of the operator replaces the semantic value of the identifier on the right. This value must be equal in type to the declared type of the identifier.
+ =	The add AND assignment operator is the two characters +=. This operator adds the semantic value of the operand on the right to the value of the left operand. This operator can only be applied to operands of primitive data types.
+-	The subtract AND assignment operator is the two characters -=. The operator subtracts the semantic value of the operand on the right from the value of the left operand. This operator can only be applied to operands of primitive data types.

4.2.4 Relational Operators

= =	Two sequential equal signs denotes an equality operator. This operator checks if the values of the two operands are equal in reference. If they are equal, it returns a “true” boolean, else it returns a “false” boolean.
!=	The characters != denote a “not equal” operator. Like the equality operator, it checks to see if the two operands are equal or not. If the left and right operands are equal in reference, then the expression evaluates to “false”. If the operands are not equal in reference, then the expression evaluates to “true”.
>	The character ‘>’ denotes the “greater than” operator. This can only be used when the left and right operands are primitive types. The “greater than” operator checks if the value of the left operand is greater than the value of the right operand. If this is true, then the expression returns true. If not, the expression evaluates to false.
<	The character ‘<’ denotes the “less than” operator. This can only be used when the left and right operands are primitive types. The “less than” operator checks if the value of the left operand is less than the value of the right operand. If this is true, then the expression returns true. If not, the expression evaluates to false.
> =	The characters “>=” denote the “greater than or equal to” operator. This operator checks if the value of the left operand is greater than or equal to the value of the right operand. If this is true, then the expression returns true. If not, the expression evaluates to false. Like the “greater than” and “less than” operators, this operator can only be used when both operands are primitive types.
< =	The characters “<=” denote the “less than or equal to” operator. This operator checks if the value of the left operand is less than or equal to the value of the right operand. If this is true, then the expression returns true. If not, the expression evaluates to false. Like the “greater than”, “less than”, and “greater than or equal to” operators, this operator can only be used when both operands are primitive types.

4.2.5 Arithmetic Operators

These operators only can be performed on primitive data types.

+	The '+' character signifies the addition operator. It adds the semantic value of the operands on the left and right side.
-	The '-' character is the subtraction operator. It subtracts the value of the right hand operator from the left hand operator to produce a new result.
*	The '*' character is the multiplication operator. It multiplies the values of the operands on the left and right side of the operator.
/	The '/' character denotes the division operator. This divides the value of the left hand operator by the value of the right hand operator.
%	The '%' character is the modulus operator. It divides the value of the left hand operator by the value of the right hand operator, and returns the remainder of the division.

4.2.6 Logical Operators

&&	The characters "&&" signify the logical AND operator. This operator checks the value of two boolean operands. If both boolean operands evaluate to true, then the expression returns true. In any other case, the expression evaluates to false.
||	The characters "||" signify the logical OR operator. Like the logical AND operator, this checks the value of the two boolean operands, but it only returns true if any of the operands (the left operand, the right operand, or both operands) evaluate to true.

4.2.7 Graphical Operators

These operators only work when both operands are Nodes.

->	The characters "->" signify the directional link operator. It adds a directional link (edge) from the operand on the left hand side to the operand on the right hand side. The operand on the right hand side is added to the list of destination nodes of the operand on the right hand side.
<->	The characters "<->" denote the bidirectional link operator. It adds a bidirectional (and therefore, undirected) link between the left operand and the right operand. In each Node operand, the other Node operand is added to the list of destination nodes.

add	This adds an attribute to either a Node or Edge type. To add an attribute, a String tag, a type, and a value must be added. The syntax for doing so, given Node n, is: n add [“name”: String “John”]; n add [“height”: int 71];

4.2.8 Parenthesized Expressions

Parenthesized expressions are a tool to override the built-in order of operations in GraphQuil. Parenthesis force the evaluation of the expression inside of them before using that value in whatever other expression the parenthetical is contained in. For example, if a given expression is 1 - 2 + 3, the internal order of operations would give the equivalent of (1 - 2) + 3, which equals 2. If however, parenthesis are added to make 1 - (2 + 3), which equals -4. This works for all operators in all expressions.

4.2.9 Function Creation

Functions are declared by giving a return type of the function, a unique identifier for the function, any arguments passed in, then creating and closing a block with the function body.

The syntax is as follows:

```
        return_type identifier (argtype arg_identifier,…)        {        … return out;  }
```

The type of out must match the return type declared at the beginning of the function, and each function must return a type. The only exception to this rule is if the function is declared with the return type *void*, then the function should not return any type. If at any point the function should return before the end of the block, putting “return;” will break out of the function and return to wherever the function was called.

4.2.10 Linking Nodes

Given two nodes *node1* and *node2*, there are two main ways of creating edges between the nodes, weighted and unweighted. Edges that have no weight merely create a connection between the two nodes and contain no more information about those connections. They are created using the following statements:

```
        node1 -> node2;        #Creates an unweighted, directed edge from node1 to node2
        node1 <-> node2;       #Creates an unweighted, bidirectional edge between the
nodes
```

To create a weighted node, there must first be a specified edge type (see section 3.5 which describes how to do so). Then, the syntax requires an assignment of an EdgeType to the edge created between the two nodes. In the same way a variable is assigned a value, the edge created between two nodes can be assigned a value (which requires a type) by assigning a newly constructed weighted edge to the connection between two nodes, as seen here:

```
        EdgeType relationship has (String type, int closeness, bool positive);
```

```
billy -> allie = new relationship("siblings", 10, true);
```

Finally, either kind of edge can only be valid if both nodes are in the same graph. The compiler will give an error if the two nodes are not in the same graph.

4.2.11 Accessing Data from Non-primitive Types

To access the data stored in a non-primitive type, the operator '.' is used, followed by the identifier given to the piece of data you want to access. This access has highest precedence in the order of operations, and the value will be pulled from the object before any other operations are done using the value, unless overridden by parenthesis. It works for accessing the values in nodes, edges, the length of arrays, and the node array of a graph. All of the following are valid:

```
int x = john.age;            #where john is a Node containing an int named age
x = intArray.length;
Node[] nodes = g1.nodes;            #note that this returns all nodes in a given
graph
String str = edge1.relationship;#where edge1 is an Edge containing said String
```

4.2.12 Using Edges to Retrieve Data

Accessing all the nodes connection to a node is done through the edges directed outward from a node. To access these, each node type has an implicit array stored inside it with the identifier *edges*. This array contains values of type Edge (which includes all EdgeTypes and unweighted edges), and can be accessed in the same way any other data can be accessed from a node by the following:

```
Edge[] edgeArray = node_indentifier.edges;
```

Then, that array can be used to iterate through and find any particular edge or node needed. Additionally, each edge, whether weighted or unweighted, contains a reference to the destination of the edge, associated with the identifier *dest*. This is done using the same syntax for accessing data for any non-primitive type, as follows: Node example = edge1.dest;

4.2.13 Accessing Data from Nodes and Edges

To access data within a Node or Edge that has been added to it, the same String tag used to add the data must be used. After ["tag", type x] has been added to a Node or Edge, accessing the data requires using that same tag and returns the given type. For example:

```
int out;
Node n1;
n1 add ["height": int 71];
out = n1["height"];
```

The access token evaluates as an expression to the type associated with that tag used in the add method.

4.3 Statements

At their core, statements execute specific tasks, control the order and flow of processes being taken, and are performed in the order in which they appear.

4.3.1 Function Calls

Function call statements are performed simply by using the desired function in any kind of expression. The function call takes highest precedence and will return its data type and use that piece of data in the expression. See the following example:

```
    String s; s = getUni(name);
```

Where getUni is defined as: String getUni(String name) { ... }

4.3.2 Conditional Statements

Usage of the *if* statement allows a certain set of steps to be taken given an explicit condition.

For example, here is the construction for an *if* statement based on an explicit condition.

```
    if (node1.year== 2015) {
        #insert set of tasks to be performed here
    }
```

Additionally, if a given condition is not met and the user desires to outline another condition, the *else if* construction can be used.

```
    if (node1.year== 2015) {
        #insert set of tasks to be performed here
    } else if (node1.year==2016) {
        #insert set of tasks to be performed here
    }
```

Lastly, the *else* construction allows users to perform a set of tasks if none of the prior mentioned *if* or *else if* conditions are met.

```
    if (node1.year== 2015) {
        #insert set of tasks to be performed here
    } else if (node1.year==2016) {
        #insert set of tasks to be performed here
    } else {
        #insert set of tasks to be performed here
    }
```

4.3.3 Iteration Statements

Iteration statements are performed using *for* or *while,* in order to generate loops that execute a set of tasks.

For example, running a set of tasks on all pieces of data in an array uses a *for* loop. The syntax is simple, where each object of the type in the array can be named (in this case tmp), and the loop will apply the set of tasks in the brackets to each object in the array. See the following example

```
    for ( Node tmp : classroom) {  #where classroom is an array filled with nodes
        #insert set of tasks to be performed here
    }
```

Using a *while* loop is very similar. The set of task outlined within the loop are executed as long as the condition denoted in the *while* loop is met, where the condition is an expression that results in a bool. The condition is re-evaluated after each pass through the loop (given the case that the condition initially passes).

```
    while (node1.year==2015){
        #insert set of tasks to be performed here
    }
```

4.4 Scope

4.4.1 Scoping within blocks

Variables defined within blocks (separated by { and }) are only accessible within the block they are defined. For example, the following is invalid:

```
    if(x = true)    {
        String abc = “abc”;
    }
    abc.length();
```

The only exception to this blocking rule is the declaration of non-primitive variables using the *new* keyword, be they graphs, nodes, or some other type. This declares a new object that is not deleted until all references to it are removed and are taken care of by behind-the-scenes garbage collection.

4.4.2 Scoping within functions

Similar to the block scoping, variables defined within a function are only applicable within that function. This applies both to the variables passed into a function and the ones named and created within it. The same exception is the declaration of non-primitives using the *new* keyword applies as before.

The main difference in the difference between function scoping and scoping within a block is in the return statement. When returning a primitive type, the value of the primitive being returned is copied directly to the expression that called the function. However, when returning a non-primitive type, a reference to the object is returned and used in the expression. The memory used to store the object remains valid until all references to it are gone, then the built-in garbage collection system removes it from the allocated memory.

5 Library Functions

Print
GraphQuil includes a built-in print to console feature. To print a String or primitive type *toPrint*, the statement:

```
print(toPrint);
```

Multiple Strings or primitives may be printed to console in one statement, separated by the plus sign as follows (for String or primitives *toPrint1* and *toPrint2*):

```
print(toPrint1 + toPrint2);
```

Any number of Strings or primitives may be printed in a single print statement as long as they are separated by plus signs as shown above.
Attempting to print the literal "\n" produces a newline in the console.
Only Strings and primitives may be printed; printing non-primitives directly is not allowed and elicits a compiler warning.

6 Project Plan

One of most important preliminary steps was determining group member availability. After figuring out when all members of the team were available during the week given their schedules, the team instituted a regular meeting time for group work and also for meeting with the TA. These preparatory housekeeping items were the backbone to being able to focus on the content and code of the project.

Our team met on Thursdays and Sundays. The TA meeting was also held on Thursdays, which allowed our group to work right after meeting with her and hearing her input. More frequent meetings were scheduled as necessary, based on deadlines and availability.

Additionally, if there were times when the entire team could not meet, we would attempt to have at least three members of the team meet so that progress could be made.

6.1 Planning and Specification Process

This phase of the project was naturally one of the first discussions to be had. The team met as a group in which we all discussed our coding backgrounds, computer science strengths, and personal interests. Leveraging this information, we were able to determine the overlaps in ideas and develop a framework for our intended language.

Then, a roadmap for the project was constructed in terms of deadlines and deliverables. This was done with the help of the TA, who provided us a set of dates and checkpoints to stay on schedule. This allowed us to strategically plan out what we needed to work on with set dates in mind.

With a team size of four, meeting as a team for the specification portion of the process was tremendously pivotal. We did not want to make significant progress to only discover that one of our teammates did not agree with one of the fundamental choices for the project. Thus, all decisions made were based on group consensus. This was the best approach, which allowed the team to view issues or certain topics from different angles, weighing the advantages and disadvantages.

While Jon Paul (Manager) led the project planning and John Heizelman (Language Guru), lead the specification process, this was a dynamic process in which all group members were involved and had equal say. For example, John Heizelman would simply raise the specification questions that needed to be addressed, provide his own opinions, and then open it up to discussion for the rest of the group. This was rather valuable as his statements were backed up by his technical prowess, but it also allowed other members of the group to provide input and demonstrate their own constructive thoughts that would push the project forward.

The specification process led to the production of our Language Reference Manual (LRM). This document fully described the motivation, intended design, explicit characteristics, and desired functionality of GraphQuil.

6.2 Development and Testing Process

Our team leveraged several tools for the development and testing phase of the process. The most pivotal was GitHub, which allowed us to collaborate and contribute simultaneously. Additionally, GitHub had the added benefit of version control, which allowed us to revert to past forms of code files in case changed needed to be reversed. In general, each member would download the last working version of the code being focused on and attempt to implement improvements on their own personal machine. Team members would only commit once those improvements were functional. This helped reduce the number of commits and dysfunctional code on the repository. However, this was unavoidable in some instances (for example, code being partionally functionally and someone uploading to get help).

The development process began after the due date of the LRM on October 27th. The files associated with development are:

- Scanner.mll
- GraphQuil.ml
- Ast.ml
- Sast.ml
- Parser.mly
- Javagen.ml
- Semantic_check.ml
- SymbolTable.ml

The testing process began after significant process was made with developing the files mentioned above. While testing was also done concurrently with development when possible, explicit testing was started leading into Thanksgiving, the week of the 17th and the 24th of November.

- Makefile
- Testall.sh
- Individual tests in folder "tests"

6.3 Style Guide

There were a number of standards and measures of coding etiquette that our team strived to follow. These were discussed during our meetings and helped us build off of each other's work. The list below is not comprehensive, but there are the ones we believed to be some of the most important and were incredibly helpful in the project.

- Incorporate comments if the purpose of a line or multiple lines isn't obvious. However, use discretion when decided whether a comment is necessary as excessive commenting is equally of no benefit.

- Follow proper indentation, an entire block of code should not be all left aligned. Ident code beneath functions, under statement constructions (e.g. loops), etc.
- Avoid incredibly lengthy lines of code. A user should not have to scroll to see where a line of code ends. This falls in the ballpark of maintaining line lengths less than 80-100 characters.
- When possible, create helper methods as opposed to large functions that accomplish a given goal. These helper methods are valuable as they can be incorporated in other functions as well.
- Functions should be distinguishable by name and should follow typical protocol learned in class, with words separated by an underscore (e.g. "let rec this_does_something here="

6.4 Project Timeline

Date	**Objective**
September 17th	Finish collaborating on initial ideas and concepts for the language, intended goals, and unique problem set solves.
September 24th	Complete Proposal
Week of October 13th	Discuss language specification, begin development
Week of October 20th	Discuss language specification, begin development
October 27th	Complete LRM
November 10th	Complete Parser, Scanner
November 24th	Complete Semantic Check
December 8th	Finish Testing, Complete Code Generation
Week of December 15th	Final Project and Presentations

6.5 Roles and Responsibilities

Here are the agreed upon roles for each member of the team.

John Heizelman –Language Guru
Jon Paul- Manager
Gemma Ragozzine –Tester
Steven Weiner- Systems Architect

While each teammate had his or her prescribed role, members were expected to contribute to all facets of the project.

Below, is a description of the leaders for each aspect of the project components generated. These members steered the ship for the described component and they decided the format and implementation to be used.

File	Component Leaders
Scanner.mll	John H/Jon P/Gemma/Steven
Language Reference Manual	John H/Jon P/ Gemma/Steven
GraphQuil.ml	Gemma
Ast.ml	Gemma/John H
Sast.ml	Gemma/John H
Parser.mly	Gemma/John H
Final Report	Gemma/Jon P
Javagen.ml	Jon P/Steven
Semantic_check.ml	Gemma/John H
SymbolTable.ml	Gemma/John H
Makefile	Gemma
Tests/ Testall.sh	Gemma

6.6 Software Development Environment

Graphquil was primarily developed within the MAC OS X environment, using a combination of tools such at Sublime, Github, and Terminal. Needless to say, OCaml, OCamllex, and Ocamlyacc were key components of the development and testing process. Scripts and Makefiles were used to automate the testing of the project components.

6.7 Project Log

```
Gemma-Ragozzines-MacBook-Pro:GraphQuil gemmaragozzine$ git log
commit 14df6d7e7420735319cf7614b535c1c8140bd247
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Wed Dec 17 17:54:19 2014 -0500

    Fixed some testing bugs, added helloworld example

commit e34fff4db0603aac098a2b06aa4bd878ed95d521
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Wed Dec 17 15:46:07 2014 -0500

    end to end testing complete and working


commit 4b7cbd659d9adae45c50a94fc23482f119d23508
Merge: 6b10f72 c55c827
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Wed Dec 17 08:26:28 2014 -0500

    Merge branch 'working' of https://github.com/jrheizelman/GraphQuil into working

commit 6b10f729dd3ff949cf6bce749cc710c88aaedb5f
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Wed Dec 17 08:25:59 2014 -0500

    bugfixes

commit c55c827e3d3ebbf1214b413004bafa03d4feefd8
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Wed Dec 17 08:24:38 2014 -0500

    trying to print executed code
```

```
commit 7a9d9abd1854eacf2e45113b1dabfee14a2bfa91
Author: Jon Paul <jpaul3144@gmail.com>
Date:   Wed Dec 17 07:10:16 2014 -0500

    final commit

commit 9e307350c5ee651a38dc1ea07ebbf7ceade54784
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Wed Dec 17 06:55:45 2014 -0500

    new branch

commit 912e66090f9b823e2c3899923e247abafec8447d
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Tue Dec 16 17:41:33 2014 -0500

    bug fixes

commit 675ba484660661bf53da29586340725572353c47
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Tue Dec 16 17:34:22 2014 -0500

    updated to be compilable

commit 915cd3e7c70678e48c7fddfb9e8d0e7881c2240a
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Tue Dec 16 17:30:29 2014 -0500

    more code gen

commit 88af74d30138de4188539676bc03e6f524fb8f27
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Tue Dec 16 06:21:48 2014 -0500

    Made significant progress on attribute checking, committing before major changes

commit 7006ba4596e8722d9c9900aed05f61c6c888386e
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Dec 15 05:32:47 2014 -0500

    Got access through parser/ast, still need to implement map and type-checking of
access
```

```
commit ab81997b74ae37b7042a76b243b7e0f37ece8f9b
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Dec 15 05:09:33 2014 -0500

    Fixed up git merge craziness

commit 7aaa52a7fe8229324a374bb8909e5446d71f3326
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Dec 15 05:05:26 2014 -0500

    Revert "Got assignment and creation of attributes through semantic check"

    This reverts commit 94db1a298cc6608d0f8a1ed7b811e52554179f66.

 Conflicts:
        Makefile
        graphQuil.ml

commit 6aac65df500a2638af2d31d944197bb984273254
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Dec 15 05:02:21 2014 -0500

    Revert "Got assignment and creation of attributes through semantic check"

    This reverts commit 94db1a298cc6608d0f8a1ed7b811e52554179f66.

    Conflicts:
        Makefile
        graphQuil.ml

commit 163522b61190155722b599b24c5813b4dda2fbe4
Merge: 94db1a2 a314ccc
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Dec 15 04:59:27 2014 -0500

    merged with pull

commit 94db1a298cc6608d0f8a1ed7b811e52554179f66
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Dec 15 04:58:19 2014 -0500

    Got assignment and creation of attributes through semantic check

commit a314ccc67d218836c2fc67b4459bddf3c0e44eef
Author: Steven Weiner <srw2163@columbia.edu>
```

```
Date:   Sun Dec 14 20:31:05 2014 -0500

        modified graphQuil.ml to pass javagen checked

        still compiles

commit b04784acf09c13cd7db1b8a7ad895bdf10d196f1
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Sun Dec 14 20:28:00 2014 -0500

        got everything to compile with javagen.ml

        changed graphQuil.ml to use javagen.ml and modified javagen.ml to compile
        properly

commit 4e66411d3dc25c9d663ff2f68200b546deba601f
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Sun Dec 14 15:52:13 2014 -0500

        fixed bugs in produceJava.ml so will compile

        new error compiling graphQuil.ml

commit 61b104dbc2ac1a07d91c97539a09cabdb13a0fb1
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Sun Dec 14 15:39:45 2014 -0500

  debugging

commit 345ae4230dffe718d0413ee87cee94793bfd10f4
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Sun Dec 14 14:11:19 2014 -0500

        fixed type mismatch in graphQuil.ml

commit 5be99cfbd4f3174f743b3d48a7d561da1dce96f5
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Dec 14 08:06:54 2014 -0500

        Finished up attribute assigns through semantic checking

commit 542e56377b171cad1721acdadca1718c4503200c
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Dec 14 05:16:33 2014 -0500
```

```
    Attributes and add through semantic check

commit 26c62991c76cdbda82885a7ec397a33c9413713c
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Sat Dec 13 17:40:08 2014 -0500

        added compiling functionality, modified makefile

    produceJava not compiling

commit 3d11f3055fc039c173051351626d2a473d22d9a3
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sat Dec 13 17:03:13 2014 -0500

        Fixed graphquil.ml errors by taking out intermediate stuff

commit d55a77daf8eef1becfefe53e86e3408cb5322afb
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sat Dec 13 16:39:09 2014 -0500

        Fixed uop type error

commit d43dae48dcec57b6a21d170c8a5deb7fe320fcf9
Merge: 177a194 f32590c
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sat Dec 13 16:23:40 2014 -0500

    attempting to commit without merge conflicts

commit 177a1941df69e030b85b10a0d275187a8c5916bd
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sat Dec 13 16:17:00 2014 -0500

        added print sast functionality

commit f32590c4c4bf2e6756e833e9c35fe27114c21cae
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Sat Dec 13 07:50:19 2014 -0500

        produce java as java generating file

commit ad8437fe219362eed453a955734eb66d7689547f
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Sat Dec 13 05:35:47 2014 -0500
```

```
            checking code gen

commit 7f1780d8398b28cf4fe6ed6840d3cc077edeb4c0
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sat Dec 13 05:14:24 2014 -0500

    Commented out int_at stuff for testing

commit db3fa96bc26de3324736759372c229a2c84ab928
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Sat Dec 13 05:06:32 2014 -0500

            checking code gen

commit b3eff594eaefa29b556889344ff22889b31e819e
Merge: 1bd0394 5ae36cc
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Sat Dec 13 03:50:34 2014 -0500

            java decs

commit 1bd0394a630ebb30b3d2090ce71df81929c1a6cb
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Sat Dec 13 03:48:49 2014 -0500

            java decs

commit 5ae36ccd0984b3882a40452c63d450525631ef5f
Merge: 18068ee 11f78bc
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 21:56:31 2014 -0500

            Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

commit 18068eec6506348ebc53e3ee4e46057b6d35ad92
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 21:56:19 2014 -0500

            Got through add/intattr, still have shift/reduce conflicts

commit 11f78bcfff83491bdc78969e6a09a19eb99a7546
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Fri Dec 12 21:45:40 2014 -0500

            modified intermediate.ml
            removed extra For_t in get_java_statement
            added blank string_of_intermediate function: will
            complete once structure of intermediate.ml is finalized

commit 38cbdad76e6e7e27203309fe350c1f794f2725f5
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Fri Dec 12 21:37:29 2014 -0500

            added ability to choose to print intermediate representation

    string_of_intermediate function still has to be written in intermediate.ml

commit d4b2091dd71561c52f45c7da98ba68a982bdad8c
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 21:34:42 2014 -0500

            Started add stuff

commit c332e8fc072e5dd683e546c85b2b9f3a231c6925
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 20:50:11 2014 -0500

            Fixed error in AST around void. Makefile bug fix.

commit f9d3b1e377c4f4f3221862b1db8d0b094b76b06a
Author: Jon Paul <jpaul@dyn-160-39-236-183.dyn.columbia.edu>
Date:   Fri Dec 12 20:30:31 2014 -0500

    additions starting with java statements

commit 83ed3df50b9a8a394e9fb162c04f9c947ad96295
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Fri Dec 12 20:01:52 2014 -0500

            changed parser and sem_check so that main method must have return type void

commit fa0568057aeadc8a9372bf30e3de76eaf2d22c17
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Fri Dec 12 16:12:59 2014 -0500

            Semantic check finished

commit fdd97bda65bda290f2221a1ad276e9466be60fc2
Author: Gemma Ragozzine <gr2390@columbia.edu>
```

```
Date:   Fri Dec 12 15:21:41 2014 -0500

        Ready and working for testing

commit 9f52dabffd7fe444388b51e6fb9d8925588efe05
Merge: 5d84b5d 99a0ad6
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Fri Dec 12 10:55:43 2014 -0500

        Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

    Conflicts:
                graphquil
                parser.cmo
                parser.ml

commit 5d84b5dff60653262a4228e3e304f75e1f6158b5
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Fri Dec 12 10:54:19 2014 -0500

        files compiling, changes to graphquil module need to be made but almost done, don't
change

commit 99a0ad6d8034c44425eb4e49d0dd2b1a5e2f635c
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 07:04:00 2014 -0500

        Cleaned up files a bit, removed accidentally included test batch

commit c797b0546dbee78b175fb5bc30cd4c3524b79626
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 07:01:18 2014 -0500

        Made progress on declaring variables inside functions, still getting shift/reduce so
commented out

commit 0b45b0a4a84e742d90b013c4d9c908665602d042
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Fri Dec 12 04:06:59 2014 -0500

        Added double literals, fixed Makefile bug

commit 91e5e2da9bda4782d3218a011ffdfb1b359a5020
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Fri Dec 12 00:44:13 2014 -0500
```

```
    Errors fixed, ready to start testing

commit b94a55f110bf5e6d400d58477bacb3cda3bf4756
Merge: 587a543 d58e19b
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 23:47:32 2014 -0500

    Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

commit 587a5432e40e3557bf03ce6b4cb0e60f3932f00a
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 23:44:56 2014 -0500

    Added better makefile, no errors up to semantic check

commit d58e19bd28c43ec52c81f2c403413425699379e4
Author: Steven Weiner <srw2163@columbia.edu>
Date:   Thu Dec 11 22:09:18 2014 -0500

    added javagen.ml file

    began actual code generation

commit 14c6b7e9691ecfb561db9b170a83e8ca7da40237
Merge: 2438f45 cff3726
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 22:03:42 2014 -0500

    Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

commit cff37262f6792c61af1e36da8a5176019bb88901
Author: Jon Paul <jpaul@dyn-160-39-236-182.dyn.columbia.edu>
Date:   Thu Dec 11 22:03:26 2014 -0500

    edits to getting string names

commit 2438f4558f82ce3ae3b118fff3a452bccdad763e
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 22:02:40 2014 -0500

    fixed all errors for parser scanner and ast

commit 6f5e271662f9d93de1c9edade7c00552d3cd26ee
Author: Jon Paul <jpaul@dyn-160-39-236-182.dyn.columbia.edu>
```

```
Date:   Thu Dec 11 20:11:02 2014 -0500

        edits to java gen

commit cc93e279b7b1d5707a4ae8ea0049dbd441e29f57
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 18:01:02 2014 -0500

        began debugging, reduce/reduce conflicts in parser.mly that still need to be figured
out

commit b15a7e634654fecce0cece1ac62877059a063f8e
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 16:14:03 2014 -0500

        fished writing semantic check, not tested yet

commit fda4e3f389dade29381e47fbcdbb9171f01ef393
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 13:33:17 2014 -0500

  continued to work on semantic check

commit c8cd615634b2863f568d43e0d4ea42068ee6a8bd
Author: Jon Paul <jpaul@dyn-160-39-236-182.dyn.columbia.edu>
Date:   Thu Dec 11 05:16:42 2014 -0500

        start of intermediate code gen

commit 0523f6ffae66db4a4a34b6ad12d0a617d1a2c13f
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Dec 11 00:27:26 2014 -0500

        revised sast, began check

commit 32344913cf88d3fe09cd33d000f7e16b5a382df0
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Wed Dec 10 23:11:25 2014 -0500

        finished symbol table

commit f635ef7dca0afa14135129453871f0b990b83504
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Wed Dec 10 20:44:51 2014 -0500
```

```
    finished code to print out AST, added block id concept, started symbol table file

commit 84949053a63d9bb9c53f0f1c072091d6db84e393
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Thu Nov 27 22:31:16 2014 -0500

    added things to sast and ast

commit bfd1cd85ec65d5dc1919d95f62eb002b466a5fa3
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Tue Nov 25 18:59:43 2014 -0500

    Work done on plane flight

commit 2166144cbdf866c45d9ab024ce2aad94da87d9d0
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Nov 24 07:29:43 2014 -0500

    Cleaned up Parser a bit, only thing left is arrays

commit 488ac7d2aaabaa182121b22f7491bf8b05fa647d
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Nov 24 06:53:28 2014 -0500

    Added a type system, cleaned up parser.

commit 5af896e6b50d9a46f0b51e7d3c9045de32967ab9
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Mon Nov 24 00:33:31 2014 -0500

    Fixed assign/vdecs in parser/ast

commit 3b845447381e28bb99de9bd86c2dc5fa71f3fb22
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 23 21:41:05 2014 -0500

    Overrode local changes that broke code

commit 312bd7b8b99e53af1525a259084141c355bc6182
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 23 21:39:37 2014 -0500

    Added make script

commit b66f6a07d81872b2bb03e463bacf8661de4e8265
```

```
Merge: 084bd92 c7f838f
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sun Nov 23 21:12:10 2014 -0500

    "After ta help"Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

commit 084bd92bc94232e11615076b6cff38db06686f50
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sun Nov 23 21:11:29 2014 -0500

        after TA help

commit c7f838f6d2b04878970d8a9246efba8c38771087
Author: srweiner <srw2163@Columbia.edu>
Date:   Fri Nov 21 16:45:41 2014 -0500

        first commit of sast.ml

commit e888ec334eb15f9fa21de2ca1c4b3c780f29da73
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Thu Nov 20 17:38:10 2014 -0500

        Removed extra stuff from print code

commit e3e08251841ba461b227b643ad4774210e8f0471
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Thu Nov 20 16:44:44 2014 -0500

        Added print ast code

commit e91e758494793ab00b369dc27d72aa8bf33179aa
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Wed Nov 12 21:16:40 2014 -0500

        scanner, parser, ast files now compile

commit 69afda91077ba749662c75e06e4b4e5b8084d3c0
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sun Nov 9 19:26:18 2014 -0500

        fixed some compilation issues

commit 7e8ca506c33e501e2e385f529cb4d79008a246a5
Author: srweiner <srw2163@columbia.edu>
Date:   Sun Nov 9 17:11:35 2014 -0500
```

```
    modified parser.mly to fix shift/reduce errors

commit c3e865936dd16ea6c97bd70747d303fd73c5cb8e
Author: srweiner <srw2163@columbia.edu>
Date:   Sun Nov 9 17:07:14 2014 -0500

    modified parser.mly file to fix error

    27 shift/reduce conflicts

commit 093aefa4e24e97c2677cf3405f3553158d9ad49d
Author: jp3144 <jpaul3144@gmail.com>
Date:   Sun Nov 9 17:06:00 2014 -0500

    Update parser.mly

commit 735d8d7c465f21698eff66f3df4e0f178e43015e
Author: jp3144 <jpaul3144@gmail.com>
Date:   Sun Nov 9 16:22:52 2014 -0500

    Update parser.mly

commit 937381cfa709639b2858d6d7b84a84103690df9d
Author: jp3144 <jpaul3144@gmail.com>
Date:   Sun Nov 9 16:15:45 2014 -0500

    Update parser.mly

commit 7e6eb79257f40bf4cfbd84d8381f987d16329290
Author: srweiner <srw2163@columbia.edu>
Date:   Sun Nov 9 15:56:50 2014 -0500

    modified expr definiion in parser.mly

    no errors given my ocamlyacc, but more features still have to be added.

commit 86eae06af2eee16a5874403bea434c08c678e671
Author: srweiner <srw2163@columbia.edu>
Date:   Sun Nov 9 15:47:46 2014 -0500

    changes made to parser.mly based on MicroC example

  GraphQuil specific changes still have to be made. Currently ocamlyacc
    gives 4 rules never reduced error.
```

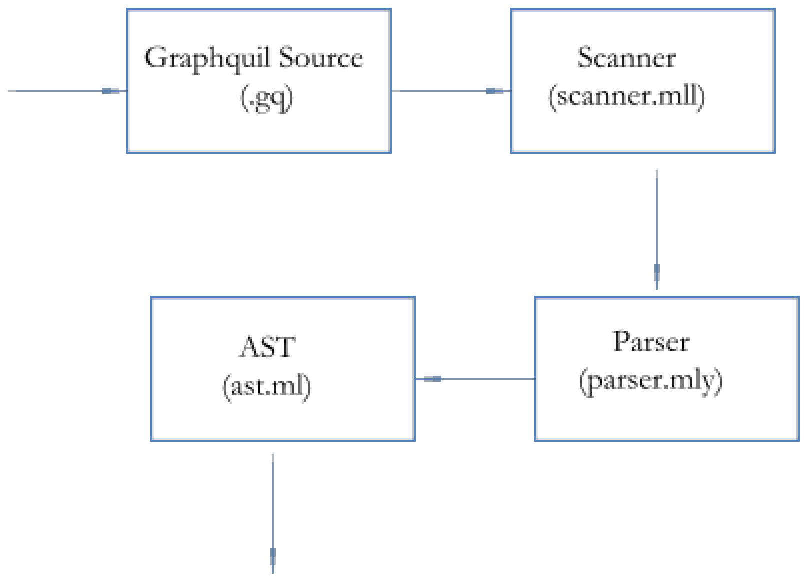
```
commit 3af21aa4f283d1308c27019184321ed9c920cf55
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 9 15:22:13 2014 -0500

    Fixed typo in parser, added colon

commit 9fa2f1111ae19f4a67e206026c61b077ae015632
Merge: 135d98d 94a4360
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 9 15:18:48 2014 -0500

    Merge scanner.ml

commit 135d98dd761f0857cc73cab913e599ad88cc617a
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 9 15:17:04 2014 -0500

    Ran ocamllex on scanner

commit 94a4360217b696455832caa9d24fc50cbb7c866c
Author: srweiner <srw2163@columbia.edu>
Date:   Sun Nov 9 15:13:29 2014 -0500

    added tokens to parser.mly

 14 rules never reduced error. tokens need to be added in grammar.

commit bd2c636b6a8e253f11aaa26d945f998b8a33daec
Author: jp3144 <jpaul3144@gmail.com>
Date:   Sun Nov 9 15:08:39 2014 -0500

    Update scanner.mll

commit bfbef7ec055998eae843ab5975d8c0c49a48f5de
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 9 14:58:33 2014 -0500

    All keywords and operators added to Scanner.mll

commit 83a9ab82de9b2dc848899ab38b1d9d386db8ae76
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 9 14:52:26 2014 -0500

    Added more operators and tokens to list
```

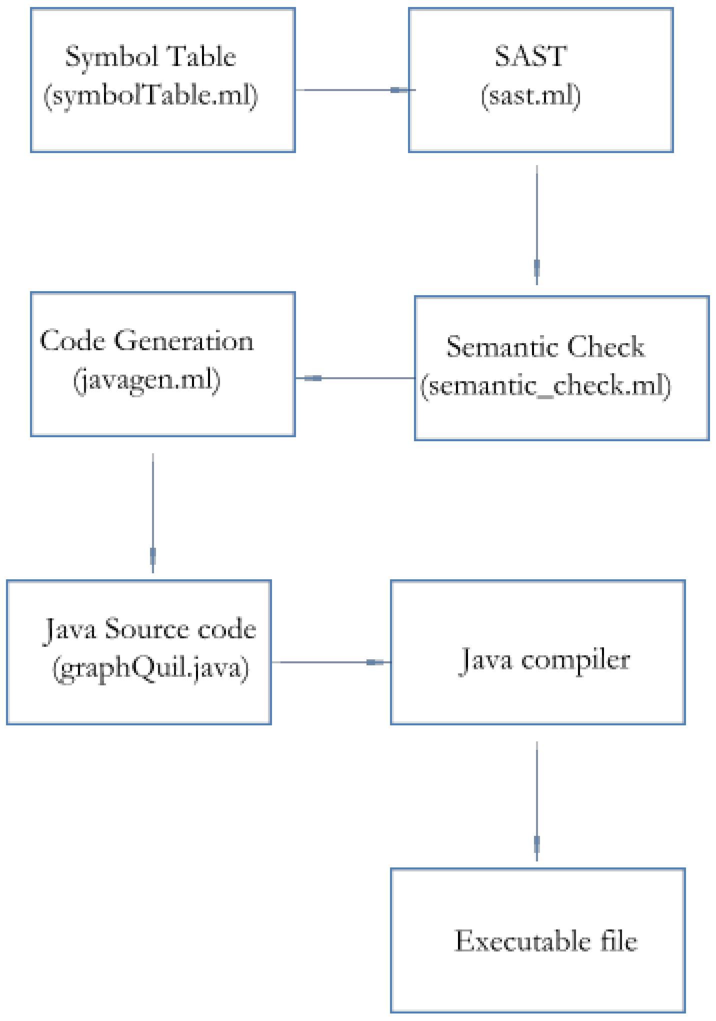
```
commit 1a83e2c2ed87d295187040dedea5aaf263168f58
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Nov 9 14:34:53 2014 -0500

        Added all keywords and organized order

commit a492511b82d2d96b07120f6f86eb5bf17440cfdb
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sun Oct 26 18:34:09 2014 -0400

        Added to scanner, parser and ast

commit 1fc4b4b866f2450e696c068d58b5cc464a5a5377
Merge: bbf2221 825e644
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sun Oct 26 18:25:28 2014 -0400

        Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

    Conflicts:
                parser.mly

commit bbf2221f1fe7307654dcad592674bdfb5dcbe1e4
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Sun Oct 26 18:18:19 2014 -0400

        Added to scanner, created preliminary parser and AST

commit 825e64429010512b5ab8cec45d0f5a3a1b9694b4
Merge: ade9272 24948fe
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Tue Oct 21 23:03:37 2014 -0400

        Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

commit 24948fea759e7a04c70ce2c5c8320c679366de30
Author: jp3144 <jpaul3144@gmail.com>
Date:   Mon Oct 20 16:40:55 2014 -0400

    "Ast.ml"

commit ade92727fad267b4497315dcfcd63907e392dfd2
Merge: ada117a d6aa5d2
Author: John Heizelman <john.heizelman@gmail.com>
```

```
Date:   Mon Oct 20 14:55:23 2014 -0400

        Merge branch 'master' of https://github.com/jrheizelman/GraphQuil

commit d6aa5d20269cff9c2b0347116ff07fac390a628d
Author: Gemma Ragozzine <gr2390@columbia.edu>
Date:   Mon Oct 20 14:17:08 2014 -0400

    "Scanner.mll"

commit ada117abb46afdd9a7f972d13625abdbd4209991
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Oct 19 22:04:14 2014 -0400

        Added parser.mly

commit e8519ad2ad271bf4495859a697ed7fabf2459720
Author: John Heizelman <john.heizelman@gmail.com>
Date:   Sun Oct 19 21:47:51 2014 -0400

        Initial commit
```

7 Architectural Design

7.1 Major Translator Components





7.2 Interfaces Between Components

The GraphQuil source file is first fed into the scanner to tokenize the input. Then, the tokens are fed into the parser to check for syntactic correctness. The AST lists all of the types in graphquil, and lays out the program structure, which is a tuple of two lists: ([variables], [functions]). The program is then semantically checked to verify that the structure is semantically correct. If the program passes semantic check, it gets generated into java btyecode, is run through the java compiler and becomes a fully executable file.

7.3 Component Implementation

See section 4.4 for a breakdown of who led the completion of which portions of the project

8 Test Plan

8.1 Sample Problem

Sample 1: helloWorld.gq:

```
/*
 * Hello World Example
 * Author: Gemma Ragozzine
 */

 void main(){
       print("hello world");
 }
```

Java Output File: graphQuil.java:

```
public class graphQuil {


  public static void main(String[] args) {
System.out.print("hello world");



}

}
```

Sample 2: test-full1.gq

```
void main()
{
  print(39 + 3);
```

Java Output File: graphQuil.java:

```
public class graphQuil {


  public static void main(String[] args) {
System.out.print(39 + 3);
}
}
```

Sample 3: test-parser3.gq

```
void main()
{
	int a;
	String b;
	while(true) {
		a = 1;
		b = "gemma";
		print(a);
		print(b);
	}
}
```

*warning: this creates an infinite loop during execution!

Java Output File: graphQuil.java:

```
public class graphQuil {

  public static void main(String[] args) { int  a; String  b; 
while(true) {

a = 1;
b = "gemma";
System.out.print(a);
System.out.print(b);

}}}
```

8.2 Test Suites and Explanation

At first, testing seemed like it would be a daunting process. I was surprised to discover that it turned out to be one of the most enjoyable aspects of the project for me. It was rewarding to watch the program that I put so much effort into come to life in different ways via automated and manual testing. In addition to being enjoyable, the testing process proved to be absolutely essential in getting even the smallest aspect of the project to work correctly. For example, when we were writing the parser, we thought that we were done because it compiled without any shift/reduce or reduce/reduce errors. Once we tested it however, we discovered that the parser didn't parse what we expected it to parse at all. Running test suites was extremely helpful in discovering errors in our program logic. We all found that OCaml code was difficult to visualize since we were new to the language, so the testing process was a key tool to understanding exactly how our code worked.

I created many testing options that correspond to the stages of the compiler. We have parser checks to check if a source code could be parsed correctly, a test to print out the Ast of the program, a test to print out the symbol table of the project, as well as a test to print the Sast of the project (though we found that to be redundant of the Ast and thus less useful). We also found it extremely useful to purposefully feed source code into these that we expected to fail. This ensured that the program accepted and rejected what we wanted and acted according to our logic.

- Gemma
Tester

8.3 Testing Automation

Testing automation is done via the shell script, testall.sh. It runs through the list of source code (*.gq) examples that are included in the file /tests and compares the output to the expected output. This made the testing process quicker and enabled us to accurately test after making small changes to the code to ensure that present changes did not affect modules already built.

8.4 Roles

All of the source code for testing, along with the automated testing script and print functions embedded in the OCaml modules that assisted testing were completed by Gemma Ragozzine, Tester.

9 Lessons Learned

9.1 Gemma

The majority of GraphQuil's shortcomings are due to lack of time, so I learned the lesson of starting early. To be more precise, I learned the lesson of listening to peers, the professor, and my TAs when they tell me that I should start early. One reason that I did not start earlier was because I was intimidated by the project and optimistically thought that as the semester progressed I would gain more confidence and insight of how to go about it. This did not happen. I think that the biggest piece of advice to give future students is that it really does not get easier until you sit down and struggle with it for *at least* 25 hours. I literally think that I spent more time staring at online OCaml tutorials, sitting in TA office hours and spending hours going through past projects line by line, than I had to actually write my own compiler. Don't make the same mistake and leave yourself enough time to really struggle with the material so that your "Aha!" moment won't come too late, as mine did. Another piece of advice is to not rely on your team members too much. A large aspect of this project is learning how to tackle a huge software engineering feat while dealing with others. If the team divvies the project responsibilities and you realize that some people are really not making timely progress, you have to deal with it yourself and nip the problem in the bud before it turns into a huge problem.

9.2 John H

The biggest takeaway for me was the importance of sticking to deadlines. We got guidance from Prof. Edwards and our TA for when different things should be done, but for most of the semester we felt ok being a week or two behind when we should've spent the time cramming in work earlier on instead of postponing the work. Initially, it started off just taking too long with the Scanner, Parser, and AST, but then each deadline just seemed to add another week behind to what we had done and where we needed to be. Over Thanksgiving break, I was able to cram in a bunch of work and made significant progress on the Scanning and Parsing, but we got to the point where the rest of the project was so far behind that the progress being made didn't really translate into benefits for the code gen/end-of-the-road progress. I think the advice that Prof. Edwards gave about getting the simplest possible iteration of the language through the whole process then worry about adding in functionality was crucial, and we were amiss in not taking it to heart.

I think the main reason I was personally hesitant in moving forward with the whole project was that I felt comfortable with the grasp I had on scanning and parsing but lacked a good idea of what OCaml was and what semantic check really had to do. I was a bit scared of moving forward and didn't have a great idea of what to do, so initially I just kept working and adding to things we already had rather than digging into old projects and trying to get a sense for what needed to be done. By the end of the semester, OCaml and the actual functional programming stuff done in the symbolic table, semantic check, codegen, etc. really clicked for me, but it was too late for the understanding to really be translated into significant progress on the language. Even the things I did manage to get through semantic check were done without enough time remaining to get them through code gen, so there was little impact on the end result of the project from a lot of the work I completed, which is not only upsetting from a time-wasting perspective but also just didn't really help us get closer to where we could've been at the end.

Beyond those code and software project specific things, I think I also learned a good amount about how to deal with people in a group setting. I've always been the type of person to get frustrated by having to deal with other people and resort to doing everything myself, but I realized in this project that it just wasn't going to happen. Early on, I think a lot of the communication problems I had early on originated from my lack of reaching out because I didn't have enough confidence in the other group members. However, I think a large portion of that was not really giving people room to contribute and my natural instinct to take control. When we started working on putting together the parsing with semantic check, I started working more closely with Gemma, and the two of us really started to get things done more quickly than each of us trying to do everything ourselves. We had both been doing a lot of work individually, but it had essentially turned into us writing bits of code that didn't work together and didn't interact with each other well enough. Because the two of us didn't communicate about the work we were both doing individually, I think we kind of fell behind where we could've been if the two of us had started collaborating earlier in the process. By the end, it worked out with Gemma and I working together with Steven and Jon Paul working on the codegen, but the breakdown of collaborative effort early on put us in a pretty big hole. In retrospect, I wish we would've discovered the breakup of work that was most successful earlier on, because Gemma and I could've gotten much more done together throughout the semester if we'd have started earlier.

I think there simplest common thread was just that we were all a bit too carefree as far as letting time pass. There were a bunch of things that either caused or resulted from us taking too long to finish things, but really the only fix would've been to start working with more diligence and purpose earlier on. More than anything, keeping a good track of time was the lesson I gained from this project, obviously on top of all the compiler, OCaml, and related knowledge I got from the class and project.

9.3 Jon P

There were a number of key takeaways from this project experience.

The first, as with any project, is the importance of communication and constantly updating your team (especially if you're working on something outside of the group setting). We employed a number of communication techniques, from a google group, to a text message group chat, to commenting within code in order to keep the members of our group informed.

However, communication can break down at times (we all go to Columbia, each with our own hectic schedules) so a subsequent lesson learned is that, when it does, remember to be persistent and demand the most out of your teammates. For example, if you send out a message and no one responds, it's okay to send out another message or pick up the phone to make a few phone calls.

The second lesson learned is that OCaml can frustrate you for what seems like an eternity but don't give up! I personally found the learning curve of the language to be much steeper than anything I had ever encountered, however by sticking with it and asking questions when necessary, OCaml actually became something I very much admired.

Learn to love the code for what it's worth and it will begin to speak to you. For example, a lot of the error alerts when using the language couldn't be more confusing to the inexperienced user. However, after getting past the initial frustration and simply taking a second to understand what the language is trying to tell you, you'll be able to address the same errors with much more dexterity in the future.

The third lesson learned is to best position yourself for the long-game of the project. This might sound vague, but is actually a combination of many smaller lessons I learned throughout the course of the process. This includes, for example, keeping the most recent working version of the project saved in a folder *outside of the repository*. While GitHub is a great tool, merge conflicts, errors, and acts of pure weirdness within the repository are unavoidable. Thus, you might be working on something and all of a sudden after making a pull, the project can't even compile. Thus, whenever you have a folder containing the last recent working version (or even multiple working versions by date), you can restore yourself to peace. This is rather simple but can really make a difference.

One final lesson that relates to the big picture of the project is this: the process will throw you many curveballs and sometimes, you just have to swing with what you have. For example, there were a number of instances in which our entire project group couldn't meet outside of our regular meeting time despite an upcoming project. Towards the latter half of the project, what we began to do was break up into mini-groups to get as much work done as possible. However, during the beginning, we simply didn't do these additional meetings if all members couldn't make it which, in retrospect, was hampering to the speed of our progress.

Overall, this was one of the most educative group projects I've worked on during my experience at Columbia in regards to both programming knowledge and group dynamics. As funny as the often repeated joke in PLT class is ("When I die, I want my group project members to lower me into my grave…), it was an incredibly rewarding experience to work on this project with such an intelligent team on this highly stimulating endeavor.

9.4 Steven

The most important lesson learned is to start early and go one step at a time. Especially at the start, everything takes twice as long as initially expected. When a group of people who do not necessarily know each other that well begin to work together on such a large project it becomes difficult to set up meetings, especially when we all have changing and varying weekly schedules. Also important is dictating who exactly is doing what. If two people are working on the same feature at the same time without any communication, time is being wasted. Dictating who does what also provides each person with a sense of responsibility for specific features. The meetings should not be just to discuss how the project will work but also who is going to do what and what should be completed by the next meeting, as that would dramatically speed up development. Our main issue was having the time to complete everything at the end, but this is more because of imperfect planning than individual lack of time. I am certain that from this project we all learned the importance of meeting regularly and planning far in advance, especially with a project of this magnitude.

At first, given the abstract nature of the scanner, parser, and AST files, it was a little difficult to figure out how to check each component upon writing. For this reason, it becomes extremely tempting to simply write everything in the same form, correct or not. A much better idea is to come up with some method for testing before even adding the second component, which as well as allowing us to see if what we are doing is correct also has the psychological effect of removing uncertainty and creating confidence that we are following the correct path. OCaml did not prove to be very difficult once I actually got started with it; however, the hype caused by the uncertainty in h

ow to even bug test creates an unfortunately large bump in the initial process. However, after getting over that initial bump of figuring out how to actually test the code, coding becomes much simpler and actually pretty enjoyable. And of course, you know you are learning a language when all the error messages begin to make sense.

Of relevance as well is coding style. For a language like OCaml where it is not always immediately apparent what is going on, clear code is extremely helpful. For this reason, in writing the Javagen.ml I made sure to make the code very finely-grained in order to provide a better understanding of what each piece specifically does and allow for easy understanding of the code, as well as much easier testing and development.

Secondly, it is very important to have an understanding of every part of the project. It is necessary to know the specific use of every newly defined type when implementing code that depends on earlier type definitions. It is also vital to understand the structure of the code, as that determines what type is returned, which is necessary to know when actually using functions.

Regarding git or any source control system, there are a few best practices necessary to follow in a project like this where multiple people may be modifying the same file and the files are so interconnected. Firstly, it is important to always pull and merge changes before pushing. Failure to do so can and for us did cause numerous problems and time wasted. It also helps to pull and push frequently. Secondly, it is very helpful to always make sure to push a working version, especially on a branch on which everyone tries to keep as updated as possible. It wastes a lot of time to pull and realize that nothing compiles or breaks. I had to work many commits behind to be using a working version of the code; realized later that branching was a better idea than just resetting the head, but in any case, not following this rule can cause much grief. In a later version of the semantic check, the check started failing every test file but this was the only version of semantic check that actually had complex very GraphQuil-specific features, causing difficulty and lack of time in finishing up those features even while all other code generation works as it should. A source control system is also incredibly useful at keeping track of development and who did what.

Especially in a group situation, it is necessary for everyone to keep their heads at all times and not give in to stress. As the deadline approached, I was impressed and gratified by everyone's ability to remain focused on what to do next and how to improve the code further, up until the very end.

10 Appendix

10.1 Appendix A: Complete Code Listing

Scanner.mll

```
(*
Authors:
              Gemma Ragozzine
               Jon Paul
               John Heizelman
               Steven Weiner
*)

{ open Parser }

let digit = ['0'-'9']
let id = ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*

rule token = parse
[' ' '\t' 'r' '\n'] { token lexbuf }
| "/*" { comment lexbuf }
| '(' { LPAREN }
| '{' { LBRACE }
| '[' { LBRACK }
| ';' { SEMI }
| ':' { COLON }
| ')' { RPAREN }
| '}' { RBRACE }
| ']' { RBRACK }
| '%' { MOD }
| ',' { COMMA }
| '.' { PERIOD }
| '+' { PLUS }
| '*' { TIMES }
| "->" { LINK }
| "<->" { BILINK }
| '-' { MINUS }
| '/' { DIVIDE }
| "==" { EQ }
| '=' { ASSIGN }
| '.' {PERIOD}
| "!=" { NEQ }
| "<=" { LEQ }
| ">=" { GEQ }
| '<' { LT }
| '>' { GT }
| '!' { NOT }
| "&&" {AND}
| "||" {OR}
| "if" { IF }
| "else" { ELSE }
| "while" { WHILE }
| "for" { FOR }
| "return" { RETURN }
| "Graph" { GRAPH }
| "Node" { NODE }
| "Edge" { EDGE }
| "bool" { BOOL }
| "String" { STRING }
| "new" { NEW }
| "continue" { CONTINUE }
| "double" { DOUBLE }
| "int" { INT }
| "void" { VOID }
| "dest" { DEST }
| "edges" { EDGES }
| "static" { STATIC }
| "char" { CHAR }
| "in" { IN }
| "add" { ADD }
| ("true" | "false") as lxm      { BOOLLIT(bool_of_string lxm) }
| '\"' ([^'\"']* as lxm) '\"'   { STRINGLIT(lxm) }
| ''' [ ^'''] as ch ''' { CHARLIT(ch) }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm)}
| ['a'-'z' 'A' - 'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*"[]" as lxm { ARRID(lxm)}
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm)}
| ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { TYPEID(lxm)}
| eof { EOF }
| _ as char { raise(Failure("illegal character " ^ Char.escaped char))}

and comment = parse
"*/" { token lexbuf }
| _ { comment lexbuf }
```

Parser.mly

```
/*
Authors: Gemma Ragozzine
         John Heizelman
*/

%{ open Ast

let scope = ref 1 (*contents of scope == 1*)

let inc_block_num
 (u:unit) =
    let x = scope.contents in
    scope := x + 1; x (*set the contents of scope to x+1, increments it by 1*)


let parse_error s = (* Called by the parser function on error *)
  print_endline s;
  flush stdout

%}

%token LPAREN LBRACE SEMI COLON RPAREN RBRACE MOD COMMA EOF
%token PLUS TIMES LINK BILINK MINUS DIVIDE EQ ASSIGN PERIOD
%token NEQ LEQ GEQ LT GT NOT AND OR RBRACK LBRACK
%token IF ELSE WHILE FOR RETURN ADD
%token INTAT STRINGAT CHARAT BOOLAT NODE
%token GRAPH  BOOL STRING PRINT NEW CONTINUE DOUBLE EDGE
```

```
%token FALSE TRUE INT VOID DEST EDGES STATIC CHAR DO IN
%token <int> LITERAL
%token <bool> BOOLLIT
%token <string> ID TYPEID ARRID STRINGLIT
%token <string> CHARLIT

/* state precedence of tokens - need this to avoid shift/reduce conflicts */
/* goes from least to most important in precedence */
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN ADD
%left OR
%left AND
%left EQ NEQ
%left LEQ GEQ LT GT
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NEW
%right NOT
%right NEG
%left LPAREN RPAREN LBRACK RBRACK

%start program
%type <Ast.program> program

%%

expr:
 LITERAL                                { Literal($1) }
| CHARLIT                               { Char_e($1) }
| ID                                    { Id($1)}
| STRINGLIT                             { String_Lit($1) }
| BOOLLIT                               { Bool_Lit($1)}
| expr PLUS expr                          { Binop ($1, Add, $3) }
| expr MINUS expr                       { Binop ($1, Sub, $3) }
| expr TIMES expr                       { Binop ($1, Mult, $3) }
| expr DIVIDE expr                      { Binop ($1, Div, $3) }
| expr MOD expr                                { Binop ($1, Mod, $1) }
| expr LT expr                                   { Binop ($1, Less, $3) }
| expr GT expr                                   { Binop ($1, Greater, $3) }
| expr LEQ expr                                { Binop ($1, Leq, $3) }
| expr GEQ expr                                { Binop ($1, Geq, $3) }
| expr EQ expr                             { Binop ($1, Equal, $3) }
| expr NEQ expr                                { Binop ($1, Neq, $3) }
| expr OR expr                                   { Binop ($1, Or, $3) }
| expr AND expr                                { Binop ($1, And, $3) }
| expr LINK expr                        { Link($1, $3) }
| expr BILINK expr                      { Bilink($1, $3) }
| ID ADD attribute                      { Add_at($1, $3) }
| NOT expr                                    { Unop(Not, $2) }
| MINUS expr %prec NEG                  { Unop(Neg, $2) }
| expr ASSIGN expr                      { Assign($1, $3) }
| ID LPAREN actuals_opt RPAREN          { Call($1, $3) }
| LPAREN expr RPAREN                    { $2 }
```

```
/*| expr ASSIGN attribute                  { Assign_at($1, $3) }*/
| ID LBRACK STRINGLIT RBRACK           { Access($1, $3) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

any_type:
INT        { Int }
| CHAR     { Char }
| STRING   { String }
| BOOL     { Bool }
| NODE     { Node([]) }
| EDGE     { Edge([]) }
| GRAPH    { Graph }

attribute:
LBRACK STRINGLIT COLON any_type expr RBRACK { Attr($2, $4, $5) }

actuals_opt:
  /* nothing */  { [] }
  | actuals_list { List.rev $1 }

actuals_list:
  expr                      { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }

formals_list:
    vdecl                    { [$1]}
  | formals_list COMMA vdecl { $3 :: $1 }

formals_opt:
  /* nothing */  { [] }
  | formals_list { List.rev $1 }

stmt_list:
  /* nothing */    { [] }
  | stmt_list stmt { $2 :: $1 }

block:
  LBRACE vdecl_list stmt_list RBRACE { {locals = List.rev $2; statements =
List.rev $3; block_num = inc_block_num()} }

stmt:
  block                                                    { Block($1)}
  /*| vdecl SEMI                                             { Vdecl(snd
$1, fst $1)}*/
  | expr SEMI                                              { Expr($1) }
  | RETURN expr SEMI                                       { Return($2) }
  | IF LPAREN expr RPAREN block %prec NOELSE               { If($3, $5,
{locals = []; statements = []; block_num = scope.contents}) }
  | IF LPAREN expr RPAREN block ELSE block                 { If ($3, $5,
$7) }
```

```
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN block  { For($3, $5,
$7, $9) }
  | WHILE LPAREN expr RPAREN block                                { While($3,
$5) }

vdecl:
  any_type ID {$2, $1}

glb_vdecl:
  vdecl SEMI { $1 }


vdecl_list:
  /* nothing */           { [] }
  | vdecl_list vdecl SEMI { $2 :: $1 }

fdecl:
   any_type ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { fname = $2;
         formals = $4;
         body_block = {locals = List.rev $7; statements = List.rev $8; block_num
= scope.contents} ;
         ret = $1 } }
  | VOID ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { fname = $2;
         formals = $4;
         body_block = {locals = List.rev $7; statements = List.rev $8; block_num
= scope.contents} ;
         ret = Void } }

program:
       /* nothing */   { [], [] }
       | program glb_vdecl { ($2 :: fst $1), snd $1 }
       | program fdecl { fst $1, ($2 :: snd $1) }
```

Ast.ml

```
(*
Authors: Gemma Ragozzine
         John Heizelman
*)

type bop = Add | Sub | Mult | Div | Equal | And | Neq | Mod | Leq | Geq |
Greater | Less | Or

type uop = Neg | Not

type validtype = Int | Char | String | Double | Bool | Arr | Node | Edge | Graph
| Void | String_at | Int_at | Char_at | Bool_at

type attribute =
Char_rat of string * string
| String_rat of string * string
| Int_rat of string * int
| Bool_rat of string * bool
```

```
type expr=
Literal of int
| Noexpr
| Id of string
| String_Lit of string
| Binop of expr * bop * expr
| Unop of uop * expr
| Call of string * expr list
| Char_e of string
| Assign of expr * expr
| Bool_Lit of bool
| Add_at of expr * expr
| Assign_at of expr * attribute
| Access of expr * string

type variable = string * validtype

type stmt =
Block of block
| Expr of expr
| Return of expr
| If of expr * block * block
| For of expr * expr * expr * block
| While of expr * block
(*| Vdecl of  validtype * string*)
(*| Link of expr * expr*)

and block = {
  locals : variable list;
  statements: stmt list;
  block_num: int;
}


type func_decl = {
       fname : string;
  formals : variable list;
  body_block : block;
  ret : validtype
}

type program = variable list * func_decl list

(*
To be used to log tags on edges/nodes in symbol table)
string = name of attribute
validtype = type
string = tag
*)
type tag_table_entry = string * attribute

(*
To be used to log the variable in the symbol table.
```

```
string = name
validtype = type
int = block number (scope)
*)
type symbol_table_var = string * validtype * int

(*
To be used to log the variable in the symbol table.
string = name
validtype = return type
validtype list = formals list
int = block number (scope)
*)
type symbol_table_func = string * validtype * validtype list * int

(*
General declaration of either variable or function to be written in symbol table
*)
type declaration =
    SymbTable_Var of symbol_table_var
  | SymbTable_Func of symbol_table_func

(*************************
**** PRINT AST **********
*************************)

let string_of_bop = function
     Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "mod"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_unop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Literal(n) -> string_of_int n
  | Char_e(n) -> "\'" ^ n ^"\'"
  | Id(s) -> s
  | String_Lit(s) -> s
  | Bool_Lit(l) -> string_of_bool l
  | Binop(e1, op, e2) ->
      string_of_expr e1 ^ " " ^
      string_of_bop op  ^ " " ^
      string_of_expr e2
  | Unop(op, e) ->
      string_of_unop op ^ " " ^
      string_of_expr e
  | Assign(v, e) ->
      string_of_expr v  ^ " = " ^
      string_of_expr e
  | Call(f, argl) ->
    f ^ "(" ^ String.concat ", " (List.map string_of_expr argl) ^ ")"
  | Noexpr -> ""
  | Add_at(e1, e2) -> string_of_expr e1 ^ " add " ^ string_of_expr e2
  | Assign_at(e, at) -> string_of_expr e ^ " = " ^ string_of_attribute at
  | Access(e, t) -> string_of_expr e ^ "[\"" ^ t ^ "\"]"

and string_of_valid_type = function
    Int -> "int"
  | Char -> "char"
  | String -> "String"
  | Double -> "double"
  | Bool -> "bool"
  | Arr -> "array"
  | Node -> "Node"
  | Edge -> "Edge"
  | Graph -> "Graph"
  | Void -> "Void"
  | Int_at -> "int_at"
  | String_at -> "String_at"
  | Char_at -> "char_at"
  | Bool_at -> "bool_at"

 and string_of_attribute = function
 Char_rat(t, v) -> "[\"" ^ t ^ "\": " ^ v ^ "]"
| String_rat(t, v) -> "[\"" ^ t ^ "\": " ^ v ^ "]"
| Int_rat(t, v) -> "[\"" ^ t ^ "\": " ^ string_of_int v ^ "]"
| Bool_rat(t, v) -> "[\"" ^ t ^ "\": " ^ string_of_bool v ^ "]"

 let string_of_variable v = string_of_valid_type (snd v) ^ " " ^  fst v

 let rec string_of_stmt = function
   Block(b) -> string_of_block b
  | Expr(expr) -> string_of_expr expr ^ ";\n"
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e1, b1, b2) ->
      (match b2.statements with
        [] -> "if (" ^ string_of_expr e1 ^ ")\n" ^ string_of_block b1
      | _  -> "if (" ^ string_of_expr e1 ^ ")\n" ^
              string_of_block b1 ^ "else\n" ^ string_of_block b2)
  | For(a1, e, a2, b) ->
      "for (" ^ string_of_expr a1 ^ "; " ^
                string_of_expr e ^ "; " ^
                string_of_expr a2 ^ ") {\n" ^
                string_of_block b ^ "}"
  | While(e, b) ->
      "while (" ^ string_of_expr e ^ ") {\n" ^
      string_of_block b ^ "}"
  (*| Vdecl(t, id) ->
      string_of_valid_type t ^ " " ^ id ^ ";"*)

  and string_of_block (b:block) = "{\n" ^
    String.concat ";\n" (List.map string_of_variable b.locals) ^ (if
(List.length b.locals) > 0 then ";\n" else "") ^
    String.concat "" (List.map string_of_stmt b.statements) ^
    "}\n"

  let string_of_func fdecl = (string_of_valid_type fdecl.ret) ^ " " ^
    fdecl.fname ^ "(" ^ String.concat ", " (List.map string_of_variable
fdecl.formals) ^ ")\n" ^
    (string_of_block fdecl.body_block)

  let string_of_prog prog = String.concat ";\n" (List.map string_of_variable
(fst prog)) ^
    (if (List.length (fst prog)) > 0 then ";\n" else "") ^
    String.concat "\n" (List.map string_of_func (snd prog))
```

SymbolTable.ml

```
(*
Authors: Gemma Ragozzine
             John Heizelman
*)
open Ast
open Printf

(* Symbol table made of a map containing pairs of String: ast.decl pairs *)
(* fst of each pair is the string of decl, snd of each pair is the decl type *)

module SymbolMap = Map.Make(String)
module TagMap = Map.Make(String)

(* each index in the array refers to a block and holds within it the scope of
the parent block *)
let ancestor_scope = Array.make 1000 0


let string_of_decl = function
         SymbTable_Var(n, t, id) -> string_of_variable (n,t) ^ " scope: "
^string_of_int id
       | SymbTable_Func(n, t, f, id) -> (string_of_valid_type t) ^ " " ^
                                                                    n ^ "(" ^
                                                                    String.concat
", " (List.map string_of_valid_type f) ^ ") scope: " ^ string_of_int id

(* table == env in lorax *)
let string_of_symbol_table env =
       let symbol_list = SymbolMap.fold
             (fun f e symList -> (string_of_decl e) :: symList) (fst env) [] in
       let sortedMap = List.sort Pervasives.compare symbol_list in
```

```
	String.concat "\n" sortedMap

(* A symbol's id also refers to its scope *)
let rec symbol_table_get_id (name:string) env =
	let(table, id) = env in
		let to_find = name ^ "_" ^ (string_of_int id) in
			if SymbolMap.mem to_find table then id
			else
				if id = 0 then raise (Failure("Get id - Symbol " ^
name ^ " not declared in current scope! (Scope: " ^ string_of_int id ^ ")"))
				else symbol_table_get_id name (table,
ancestor_scope.(id))

(* Look for symbol in given scope (block id) and if not found, recursively check
all ancestor scopes*)
let rec symbol_table_find (name:string) env =
	let(table, id) = env in
		let to_find = name ^ "_" ^ (string_of_int id) in
			if SymbolMap.mem to_find table then SymbolMap.find to_find
table
			else
				if id = 0 then raise (Failure("Find name - Symbol " ^
name ^ " not declared in current scope! (Scope: " ^ string_of_int id ^ ")"))
				else symbol_table_find name (table,
ancestor_scope.(id))

let tag_table_find (var:string) (tag:string) table =
	if TagMap.mem var table then
		let l = TagMap.find var table in
			let check_for_tag = (fun (s,at) -> s = tag) in
				if List.exists check_for_tag l then List.find
check_for_tag l
			else raise(Failure(var ^ " does not have that tag associated
with it."))
	else
		raise(Failure(var ^ " does not have any tags associated with it."))

let tag_table_add (s1:string) (s2:string) table =
	let add = TagMap.find s2 table in
		ignore(TagMap.remove s2 table);
		if TagMap.mem s1 table then
			let l = TagMap.find s1 table in
				TagMap.add s1 (List.append l add) table
		else
			TagMap.add s1 add table

let tag_table_assign_at (name:string) (at:tag_table_entry) table =
	TagMap.add name (at :: []) table

let rec symbol_table_add_decl (name:string) (decl:declaration) env =
	let (table, id) = env in
		let to_find = name ^ "_" ^ (string_of_int id) in
			if SymbolMap.mem to_find table then raise(Failure("Symbol "
^ name ^ " already declared in this scope! (Scope: " ^ string_of_int id ^ ")"))
```

```
                    else ((SymbolMap.add to_find decl table), id)

(* Recursively add list of variables to symbol table *)
let rec symbol_table_add_var_list (vars:variable list) env =
        match vars with
                  [] -> env
                | (var_name, var_type) :: tail ->
                        let env = symbol_table_add_decl var_name
(SymbTable_Var(var_name, var_type, snd env)) env in
                                symbol_table_add_var_list tail env

let rec symbol_table_add_stmt_list (stmts:stmt list) env =
        match stmts with
                  [] -> env
                | head :: tail -> let env = (match head with
                          Block(b) -> symbol_table_add_block b env
                        | For(e1, e2, e3, b) -> symbol_table_add_block b env
                        | While(e, b) -> symbol_table_add_block b env
                        | If(e, b1, b2) -> let env = symbol_table_add_block b1 env
in symbol_table_add_block b2 env
                        | _ -> env) in symbol_table_add_stmt_list tail env

and symbol_table_add_block (b:block) env =
        let (table, id) = env in
                let env = symbol_table_add_var_list b.locals (table, b.block_num)
in
                        let env = symbol_table_add_stmt_list b.statements env in
                                        ancestor_scope.(b.block_num) <- id;(* Note
which scope we are putting this block into *)
                                ((fst env), id) (* Return old scope we started
int(block id) and name of last statement we added*)

let symbol_table_add_func (f:func_decl) env =
        let id = snd env in
                let args = List.map snd f.formals in (* Get the name of each formal
*)
                        let env = symbol_table_add_decl f.fname
(SymbTable_Func(f.fname, f.ret, args, id)) env in
                        let env = symbol_table_add_var_list f.formals ((fst env),
f.body_block.block_num) in
                                symbol_table_add_block f.body_block ((fst env), id)

let rec symbol_table_add_func_list (funcs:func_decl list) env =
        match funcs with
                  [] -> env
                | head :: tail -> let env = symbol_table_add_func head env in
                symbol_table_add_func_list tail env

let add_built_in_funcs env = symbol_table_add_decl "print"
(SymbTable_Func("print", Int, [], 0)) env

let symbol_table_of_prog (p:Ast.program) =
        (* Table starts off as an empty map with scope (block id) set to 0 *)
        let env = add_built_in_funcs(SymbolMap.empty, 0) in
```

```
             let env = symbol_table_add_var_list (fst p) env in
                    symbol_table_add_func_list (snd p) env

let empty_tag_map =
       TagMap.empty
```

Sast.ml

```
(*
Author: Gemma Ragozzine
        John Heizelman
*)

open Ast
(*Graphquil*)

(* Elements from AST but with typing added*)

let fst_of_three (t, _, _) = t
let snd_of_three (_, t, _) = t
let fst_of_four (t, _, _, _) = t

type expr_t =
    Literal_t of int
  | Noexpr_t
  | Id_t of validtype * string * int
  | Binop_t of validtype * expr_t * bop * expr_t
  | Unop_t of validtype  * uop * expr_t
  | Call_t of symbol_table_func * expr_t list
  | String_Lit_t of string
  | Char_t of string
  | Assign_t of validtype * expr_t * expr_t
  | Bool_Lit_t of bool
  | Add_at_t of expr_t * expr_t (* each expr_t is the id of the node and attr,
type checked *)
  | Assign_at_t of validtype * expr_t * attribute
  | Access_t of validtype * expr_t * string

type stmt_t =
    Block_t of block_t
  | Expr_t of expr_t
  | Return_t of expr_t
  | If_t of expr_t * block_t * block_t
  | For_t of expr_t * expr_t * expr_t * block_t
  | While_t of expr_t * block_t
  (*| Link_t of expr_t * expr_t*)
  (*| Vdecl_t of validtype * string * expr_t*)

and block_t = {
  locals_t : symbol_table_var list;
  statements_t : stmt_t list;
  block_num_t : int
}
```

```
type function_t = {
  fname_t: string;
       formals_t : symbol_table_var list;
       ret_t : validtype;
  body_block_t : block_t;
}

type program_t = symbol_table_var list * function_t list


(************************
**** PRINT SAST **********
************************)


let rec string_of_expr_t = function
    Literal_t(n) -> string_of_int n
  | Char_t(n) -> "\'" ^ n ^"\'"
  | Id_t(t, s, i) -> s
  | String_Lit_t(s) -> s
  | Bool_Lit_t(l) -> string_of_bool l
  | Binop_t(t, e1, op, e2) ->
      string_of_expr_t e1 ^ " " ^
      string_of_bop op  ^ " " ^
      string_of_expr_t e2
  | Unop_t(t, op, e) ->
      string_of_unop op ^ " " ^
      string_of_expr_t e
  | Assign_t(t, v, e) ->
      string_of_expr_t v  ^ " = " ^
      string_of_expr_t e
  | Call_t(f, argl) ->
    fst_of_four f ^ "(" ^ String.concat ", " (List.map string_of_expr_t argl) ^
")"
  | Noexpr_t -> ""
  | Add_at_t(n, e) -> string_of_expr_t n ^ " add " ^ string_of_expr_t e
  | Assign_at_t (t, e, a) -> string_of_expr_t e ^ " = " ^ string_of_attribute a
  | Access_t (t, e, s) -> string_of_expr_t e ^ "[\"" ^ s ^ "\"]"

  let string_of_symb_table_var v = string_of_valid_type (snd_of_three v) ^ " " ^
fst_of_three v

  let rec string_of_stmt_t = function
    Block_t(b) -> string_of_block_t b
  | Expr_t(expr) -> string_of_expr_t expr ^ ";\n"
  | Return_t(expr) -> "return " ^ string_of_expr_t expr ^ ";\n";
  | If_t(e1, b1, b2) ->
      (match b2.statements_t with
        [] -> "if (" ^ string_of_expr_t e1 ^ ")\n" ^ string_of_block_t b1
      | _  -> "if (" ^ string_of_expr_t e1 ^ ")\n" ^
              string_of_block_t b1 ^ "else\n" ^ string_of_block_t b2)
  | For_t(a1, e, a2, b) ->
      "for (" ^ string_of_expr_t a1 ^ "; " ^
                string_of_expr_t e ^ "; " ^
                string_of_expr_t a2 ^ ") {\n" ^
                string_of_block_t b ^ "}"
  | While_t(e, b) ->
      "while (" ^ string_of_expr_t e ^ ") {\n" ^
      string_of_block_t b ^ "}"
  (*| Vdecl(t, id) ->
      string_of_valid_type t ^ " " ^ id ^ ";"*)

  and string_of_block_t (b:block_t) = "{\n" ^
    String.concat ";\n" (List.map string_of_symb_table_var b.locals_t) ^ (if
(List.length b.locals_t) > 0 then ";\n" else "") ^
    String.concat "" (List.map string_of_stmt_t b.statements_t) ^
    "}\n"

  let string_of_func_t fdecl = (string_of_valid_type fdecl.ret_t) ^ " " ^
    fdecl.fname_t ^ "(" ^ String.concat ", " (List.map string_of_symb_table_var
fdecl.formals_t) ^ ")\n" ^
    (string_of_block_t fdecl.body_block_t)

  let string_of_prog_t prog = String.concat ";\n" (List.map
string_of_symb_table_var (fst prog)) ^
    (if (List.length (fst prog)) > 0 then ";\n" else "") ^
    String.concat "\n" (List.map string_of_func_t (snd prog))
```

Semantic_check.ml

```
(*
	Authors: Gemma Ragozzine
			 John Heizelman
*)

open Ast
open Sast
open SymbolTable


(* Structure the main function *)
let main_fdecl (f:function_t) =
	if f.fname_t = "main" then
		if f.ret_t = Void then
			if f.formals_t = [] then true
			else raise(Failure("Main method argument list must be
empty"))
		else raise(Failure("Main method must return void"))
	else false

(* called to get the type of an expression *)
let type_of_expr = function
	  Literal_t(i) -> Int
	| Noexpr_t -> raise (Failure("Type of expression called on noexpr
type."))
	| Id_t(t, _, _) -> t
```

```
       | Binop_t(t, _, _, _) -> t
       | Unop_t(t, _, _) -> t
       | Call_t(f, _) -> let (_,r,_,_) = f in r
       | String_Lit_t(s) -> String
       | Char_t(c) -> Char
       | Assign_t(t, _, _) -> t
       | Bool_Lit_t(b) -> Bool
       | Add_at_t(_, _) -> Node
       | Assign_at_t(t, _, _) -> t
       | Access_t(t, _, _) -> t

let type_of_attribute = function
       Char_rat(_, _) -> Char
       | String_rat(_,_) -> String_at
       | Int_rat(_, _) -> Int_at
       | Bool_rat(_, _) -> Bool_at

let access_type_of_attr = function
       Char_rat(_, _) -> Char
       | String_rat(_,_) -> String
       | Int_rat(_, _) -> Int
       | Bool_rat(_, _) -> Bool

(* Error raised for improper binary operation *)
let binop_err (t1:validtype) (t2:validtype) (op:bop) =
              raise(Failure("Operator " ^ (string_of_bop op) ^
                     " not compatible with expressions of type " ^
                     string_of_valid_type t1 ^ " and  " ^
                     string_of_valid_type t2 ^ "."))

(* Check binary operation *)
let check_binop (e1:expr_t) (e2:expr_t) (op:bop) =
       (* TODO check expressions type for null *)
       let (t1, t2) = (type_of_expr e1, type_of_expr e2) in
              (* Both are ints *)
              match (t1, t2) with (Int, Int) ->
                     (match op with
                     (Add | Sub | Mult | Div | Mod) -> Binop_t(Int, e1, op, e2)
                     | (Equal | Neq | Less | Leq | Greater | Geq) ->
Binop_t(Bool, e1, op, e2)
                     | _ -> binop_err t1 t2 op)
              (* Both are bools *)
              | (Bool, Bool) ->
                     (match op with (And | Or | Equal | Neq) ->
                            Binop_t(Bool, e1, op, e2)
                            | _ -> binop_err t1 t2 op)
              (* Both are chars *)
              | (Char, Char) ->
                     (match op with (Add | Sub) ->
                            Binop_t(Char, e1, op, e2)
                            | _ -> binop_err t1 t2 op)
              | _ -> binop_err t1 t2 op

let unop_err (t:validtype) (op:uop) =
        raise(Failure("Operator " ^ (string_of_unop op) ^
                " not compatible with expression of type " ^
                (string_of_valid_type t) ^ "."))

let check_unop (e:expr_t) (op:uop) =
        let t = type_of_expr e in
                match t with
                (* Expression is an int *)
                   Int ->
                       (match op with
                                Neg -> Unop_t(Int, op, e)
                              | _ -> unop_err t op)
                (* Expression is a bool *)
                 | Bool ->
                       (match op with
                                Not -> Unop_t(Bool, op, e)
                                | _ -> unop_err t op)
                 | _ -> unop_err t op

let assign_err (t1:validtype) (t2:validtype) =
        raise(Failure("Cannot assign expression of type " ^
                string_of_valid_type t2 ^ " to expression of type " ^
                string_of_valid_type t1 ^ "."))

let add_err (t1:validtype) (t2:validtype) =
        raise(Failure("Add must be of attribute type " ^
                        "to Node or Edge type, " ^ string_of_valid_type t1 ^ " to "
^
                        string_of_valid_type t2 ^ " was given."))

let check_assign (l:expr_t) (r:expr_t) =
        let (l_t, r_t) = (type_of_expr l, type_of_expr r) in
                  if(l_t = r_t) then Assign_t(l_t, l, r)
                  else assign_err l_t r_t

(* compare arg lists with func formal params *)
let rec compare_args formals actuals =
        match (formals, actuals) with
                ([], []) -> true
                | (head1::tail1, head2::tail2) ->
                        (match (head1, head2) with _ ->
                        (head1 = head2) && compare_args tail1 tail2)
                | _ -> false

 and check_func_call (name:string) (eList: expr_t list) env =
        let decl = symbol_table_find name env in
                let fdecl =
                        (match decl with
                              (* make sure it is a function *)
                              SymbTable_Func(f) -> f
                              | _ -> raise(Failure("Variable " ^ name ^ " is not a
function."))) in
                        let (fname, ret_type, formals, id) = fdecl in
                              let actuals = List.map type_of_expr eList in
```

```
                                  match name with
                                        "print" -> Call_t((fname, ret_type,
actuals, id), eList)
                                        | _ -> (* Check num of arguments *)
                                              if(List.length formals) =
(List.length actuals) then
                                                    (* Check type of args *)
                                                    if (compare_args formals
actuals) then Call_t(fdecl, eList)
                                                    else
raise(Failure("Function " ^ name ^
                                                    "'s argument types do not
match its formals"))
                                              else raise(Failure("Function " ^
name ^ " expected " ^
                                              (string_of_int (List.length
formals)) ^
                                              " arguments but was called with "
^
                                              (string_of_int (List.length
actuals))))

let rec check_valid_id (id_name:string) env =
      (* Check if id is in the symbol table *)
      let decl = symbol_table_find id_name env in
            (* Check if id is in the correct scope *)
            let id = symbol_table_get_id id_name env in
                  (match decl with
                        (* SymbTable_Var = name * type * id *)
                        SymbTable_Var(v) -> (snd_of_three v, fst_of_three v,
id)
                        | _ -> raise(Failure("Symbol " ^ id_name ^ " is not a
variable.")))

 (* Unclear if we need this
      This came in so much handy for me (JH), thanks*)
let rec get_left_value_of_expr (e:expr_t) =
      match e with
            Id_t(t, s, _) -> s
            | Binop_t(t, l, o, r) -> get_left_value_of_expr l
            | Unop_t(t, o, l) -> get_left_value_of_expr l
            | _ -> raise(Failure("Cannot get the left value of expression."))

and check_left_value (e:expr) env =
      match e with
            (* If left expression is an id *)
            Id(s) -> let (t, e, id) = check_valid_id s env in Id_t(t, e, id)
            (* If left expression is anything else *)
            | _ -> raise(Failure("Left hand side of assignment operator is
improper type"))

and check_attribute (a:attribute) env =
      match a with
      Char_rat(t, v) -> Char_rat(t, v)
```

```
       | String_rat(t, v) -> String_rat(t, v)
       | Int_rat(t, v) -> Int_rat(t, v)
       | Bool_rat(t, v) -> Bool_rat(t,v)

(* Did not check Array, construct, makeArr, Access*)
and check_expr (e:expr) env tagtab =
       match e with
        Literal(i) -> Literal_t(i)
        | Noexpr -> Noexpr_t
        | Id(s) -> let (t, st, id) = check_valid_id s env in Id_t(t, st, id)
        | Binop(e1, op, e2) ->
             let(ce1, ce2) = (check_expr e1 env tagtab, check_expr e2 env
tagtab) in
                     check_binop ce1 ce2 op
        | Unop(op, e) ->
             let ce = check_expr e env tagtab in
                     check_unop ce op
        | Call(n, eList) ->
             let checkedList = check_exprList eList env tagtab in
                     check_func_call n checkedList env
        | String_Lit(s) -> String_Lit_t(s)
        | Char_e(c) -> Char_t(c)
        | Assign(l, r) ->
             let checked_r = check_expr r env tagtab in
                     let checked_l = check_left_value l env in
                             check_assign checked_l checked_r
        | Bool_Lit(b) -> Bool_Lit_t(b)
        | Assign_at(l, r) ->
             let checked_l = check_left_value l env in
                     let checked_r = check_attribute r env in
                             check_assign_attribute checked_l checked_r tagtab
        | Add_at(l, r) ->
             let checked_l = check_expr l env tagtab in
                     let checked_r = check_expr r env tagtab in
                             check_add checked_l checked_r tagtab
        | Access(e, t) ->
             let checked_e = check_expr e env tagtab in
                     check_access checked_e t tagtab

and check_assign_attribute (l:expr_t) (r:attribute) tagtab =
       let(l_t, r_t) = (type_of_expr l, type_of_attribute r) in
             if(l_t = r_t) then
                     let at_name = get_left_value_of_expr l in
                             ignore(tag_table_assign_at at_name (at_name, r)
tagtab);
                             Assign_at_t(l_t, l, r)
             else assign_err l_t r_t

and check_add (l:expr_t) (r:expr_t) tagtab =
       let (l_t, r_t) = (type_of_expr l, type_of_expr r) in
             if(l_t = Node || l_t = Edge) then
                     let (l_name, r_name) = (get_left_value_of_expr l,
get_left_value_of_expr r) in
                             ignore(tag_table_add l_name r_name tagtab);
```

```
                            match r_t with
                            String_at -> Add_at_t(l, r)
                            | Int_at -> Add_at_t(l, r)
                            | Char_at -> Add_at_t(l, r)
                            | Bool_at -> Add_at_t(l, r)
                            | _ -> add_err l_t r_t
                            else add_err l_t r_t

and check_access (e:expr_t) (s:string) tagtab =
        let e_t = type_of_expr e in
                if(e_t = Node || e_t = Edge) then
                        let name = get_left_value_of_expr e in
                                let (n, at) = tag_table_find name s tagtab in
                                        Access_t(access_type_of_attr at, e, s)
                else raise(Failure("Access must be on type Node or " ^
                        "Edge, " ^ string_of_valid_type e_t ^ " given."))

and check_exprList (eList: expr list) env tagtab=
        match eList with
                 [] -> []
                | head::tail -> (check_expr head env tagtab) :: (check_exprList
tail env tagtab)

let rec check_statement (s:stmt) ret_type env tagtab (scope:int) =
        match s with
                 Block(b) ->
                        let checked_block = check_block b ret_type env tagtab scope
in
                                Block_t(checked_block)
                | Expr(e) -> Expr_t(check_expr e env tagtab)
                | Return(e) ->
                        let checked_e = check_expr e env tagtab in
                                let type_e = type_of_expr checked_e in
                                        if (type_e = ret_type) then Return_t(checked_e)
                                else raise(Failure("Function tries to return type " ^
                                        (string_of_valid_type type_e) ^ " but should
return type " ^
                                        (string_of_valid_type ret_type) ^ "."))
                | If(e, b1, b2) ->
                        let ce = check_expr e env tagtab in
                                let te = type_of_expr ce in
                                        (match te with
                                                 Bool -> If_t(ce, check_block b1
ret_type env tagtab scope, check_block b2 ret_type env tagtab scope)
                                                | _ -> raise(Failure("If statement
must evaluate on a boolean expression.")))

                | For(e1, e2, e3, b) ->
                        let(c1, c2, c3) = (check_expr e1 env tagtab, check_expr e2
env tagtab, check_expr e3 env tagtab) in
                                if (type_of_expr c2 = Bool) then
                                        (* Increment scope, check block to be valid
block *)
```

```
                                  For_t(c1, c2, c3, check_block b ret_type env
tagtab (scope))
                           else raise(Failure("For loop condition must evaluate
to a boolean expression"))
              | While(e, b) ->
                    let ce = check_expr e env tagtab in
                           if (type_of_expr ce = Bool)
                                  then While_t(ce, check_block b ret_type env
tagtab (scope))
                           else raise(Failure("While loop must evaluate on a
boolean expression"))
              (*| Vdecl(t, id) ->
                    symbol_table_add_var_list *)
              (*| Link(l, r) ->
                    let checked_l = check_expr l env in
                           let type_l = type_of_expr checked_l in
                                  let checked_r = check_expr r env in
                                         let type_r = type_of_expr checked_r in
                                                if(type_l = type_r) then
Link_t(checked_l, checked_r)
                                         else raise(Failure("Function tries to
link type " ^
                                                (string_of_valid_type type_l) ^ "
with type " ^
                                                (string_of_valid_type type_r) ^
"."))*)

and check_block (b:block) (ret_type:validtype) env tagtab (scope:int) =
       let variables = check_is_vdecl_list b.locals (fst env, b.block_num) in
              let stmts = check_stmt_list b.statements ret_type (fst env,
b.block_num) tagtab scope in
                    {locals_t = variables; statements_t = stmts; block_num_t =
b.block_num}


and check_stmt_list (s:stmt list) (ret_type:validtype) env tagtab (scope:int) =
       match s with
                [] -> []
              | head::tail ->
                    check_statement head ret_type env tagtab scope ::
check_stmt_list tail ret_type env tagtab scope

and check_is_vdecl_list (vars:variable list) env =
       match vars with
                [] -> []
              | head :: tail ->
                    let decl = symbol_table_find (fst head) env in
                           let id = symbol_table_get_id (fst head) env in
                                  match decl with
                                          SymbTable_Func(f) ->
raise(Failure("Symbol "^ (fst head) ^
                                                 "is a function, not a variable
"))
                                        | SymbTable_Var(v) ->
```

```
                                                let varType = snd_of_three v in
                                                        match varType with
                                                                validtype ->
(fst_of_three v, snd_of_three v, id) :: check_is_vdecl_list tail env


let rec check_is_fdecl (func:string) env =
        let fdecl = symbol_table_find func env in
                match fdecl with
                          SymbTable_Var(v) -> raise(Failure("Symbol is a variable,
not a function."))
                        | SymbTable_Func(f) -> f

and check_function (f:func_decl) env tagtab =
        let checked_block = check_block f.body_block f.ret env tagtab 0 in
                let checked_formals = check_is_vdecl_list f.formals (fst env,
f.body_block.block_num) in
                        let checked_scope = check_is_fdecl f.fname env in
                                {fname_t = fst_of_four checked_scope; ret_t = f.ret;
formals_t = checked_formals; body_block_t = checked_block }

and check_function_list (funcs:func_decl list) env tagtab =
        match funcs with
                  [] -> []
                | head::tail -> check_function head env tagtab ::
check_function_list tail env tagtab

and check_main_exists (f:function_t list) =
        if (List.filter main_fdecl f) = [] then false else true

and check_program (p:program) env tagtab =
        let vs = fst p in
                let fs = snd p in
                        let checked_vs = check_is_vdecl_list vs env in
                                let checked_fs = check_function_list fs env tagtab in
                                        if(check_main_exists checked_fs) then
(checked_vs, checked_fs)
                                        else raise(Failure("Function main not found."))
```

Javagen.ml

```
open Ast
open Sast
open Printf

let rec writeToFile filename pString = (* writes to file *)
  let file = open_out (filename ^ ".java") in
    fprintf file "%s" pString

and write_code filename p = (* adds class structure to java file and has the
code written *)
  let (symbolvarlist, functionlist) = p in
  let stmtString = gen_function_list functionlist and
```

```
      symbolString = gen_var_list symbolvarlist in
  let output = sprintf "
public class %s {
  %s
  %s
}
  " filename symbolString stmtString in
  writeToFile filename output ; filename

and gen_var_list varlist =
  let output = List.fold_left (fun a b -> a ^ ((gen_var b)) ^ ";") "" varlist in
  sprintf "%s\n" output

and gen_var var =
  let (name, t, scope) = var in
  sprintf "%s %s" (gen_type t) name

and gen_type = function
    Int -> " int "
  | Char -> " char "
  | String -> " String "
  | Bool -> " boolean "
  | Void -> " void "
  | _ -> " other "

and gen_function_list functionlist =
  let output = List.fold_left (fun a b -> a ^ (gen_func b)) "" functionlist in
  sprintf "%s\n" output

and gen_func func =
  let returntype = func.ret_t and
      funcname = func.fname_t and
      params = func.formals_t and
      internals = func.body_block_t in
  let helper = function
    "main" -> sprintf "public static void main(String[] args) {"
  | _ -> (
    let paramshelper para =
      let paraams = List.fold_left (fun a b -> a ^ ((gen_var b) ^ ", ")) "" para
in
      String.sub paraams 0 (String.length paraams - 2) in
    let helper2 rt fn para =
      let t = gen_type rt in
      sprintf "public %s %s (%s) {\n" t fn (paramshelper para)
    in helper2 returntype funcname params
  ) in
  let output = (helper funcname) ^ (gen_block internals) ^ "\n}\n"
  in sprintf "%s\n" output

and gen_block block =
  let vars = block.locals_t and
      stmts = block.statements_t in
  let output = (gen_var_list vars) ^ (gen_stmt_list stmts) in
  sprintf "%s\n" output
```

```
and gen_stmt_list stmts =
  let output = List.fold_left (fun a b -> a ^ (gen_stmt b)) "" stmts in
  sprintf "%s\n" output

and gen_stmt = function
    Block_t(block) -> gen_block block
  | Expr_t(expr) -> gen_expr expr
  | Return_t(toreturn) -> gen_return_stmt toreturn
  | If_t(condition, block1, block2) -> gen_if_stmt condition block1 block2
  | For_t(expr1, expr2, expr3, block) -> gen_for_stmt expr1 expr2 expr3 block
  | While_t(expr, block) -> gen_while_stmt expr block

and gen_for_stmt expr1 expr2 expr3 block =
  let output = sprintf "for(%s; %s; %s) {\n%s\n}" (gen_expr expr1) (gen_expr
expr2) (gen_expr expr3) (gen_block block) in
  sprintf "%s\n" output

and gen_while_stmt expr block =
  let output = sprintf "while(%s) {\n%s\n}" (gen_expr expr) (gen_block block) in
  sprintf "%s\n" output

and gen_return_stmt expr =
  let output = (gen_expr expr) in
  sprintf "return %s;\n" output

and gen_if_stmt condition block1 block2 =
  sprintf "if(%s) {\n%s\n}\nelse {\n%s\n}" (gen_expr condition) (gen_block
block1) (gen_block block2)

and gen_expr = function
    Literal_t(lit) -> string_of_int lit
  | Noexpr_t -> ""
  | Id_t(t, name, scope) -> sprintf "%s" name
  | Binop_t(t, expr1, op, expr2) -> gen_binop t expr1 op expr2
  | Unop_t(t, op, expr) -> ""
  | Call_t(symtablefunc, exprlist) -> gen_call_func symtablefunc exprlist
  | String_Lit_t(str) -> Char.escaped '"' ^ str ^ Char.escaped '"'
  | Char_t(str) -> sprintf "'%s'" str
  | Assign_t(t, expr1, expr2) -> gen_assign t expr1 expr2
  | Bool_Lit_t(b) -> string_of_bool b
  | Add_at_t(expr1, expr2) -> string_of_expr_t expr1 ^ " add " ^
string_of_expr_t expr2

and gen_assign t expr1 expr2 =
  let output = sprintf "%s = %s;" (gen_expr expr1) (gen_expr expr2) in
  sprintf "%s\n" output

and gen_binop datatype e1 op e2 =
  match op with
    Add -> gen_expr e1 ^ " + " ^ gen_expr e2
  | Sub -> gen_expr e1 ^ " - " ^ gen_expr e2
  | Mult -> gen_expr e1 ^ " * " ^ gen_expr e2
  | Div -> gen_expr e1 ^ " / " ^ gen_expr e2
```

```
    | Mod -> gen_expr e1 ^ " mod " ^ gen_expr e2
    | Equal -> gen_expr e1 ^ " == " ^ gen_expr e2
    | Neq -> gen_expr e1 ^ " != " ^ gen_expr e2
    | Less -> gen_expr e1 ^ " < " ^ gen_expr e2
    | Leq -> gen_expr e1 ^ " <= " ^ gen_expr e2
    | Greater -> gen_expr e1 ^ " > " ^ gen_expr e2
    | Geq -> gen_expr e1 ^ " >= " ^ gen_expr e2
    | And -> gen_expr e1 ^ " && " ^ gen_expr e2
    | Or -> gen_expr e1 ^ " || " ^ gen_expr e2
    | _ -> "other"

and gen_call_func symtablefunc exprlist =
  let (name, t, s, scope) = symtablefunc in
  let namehelper name =
    match name with
      "print" -> "System.out.print"
    |  _ -> name in
  let argshelper args =
    let aargs = List.fold_left (fun a b -> a ^ ((gen_expr b) ^ ", ")) "" args in
    String.sub aargs 0 (String.length aargs - 2) in
  let output = sprintf "%s(%s);" (namehelper name) (argshelper exprlist) in
  sprintf "%s\n" output


(*
and gen_stmt stmt = (* generates statements in java *)
  let (a,b,c) = stmt in
  match b with
    Return_t(exp) -> gen_return_stmt exp
  | _ -> sprintf "other"
(*
and gen_expr expr =
  match expr with (* generates expressions in java *)
    string -> sprintf "%s" (string_of_expr_t expr)
  | _ -> sprintf "other"

and gen_return_stmt exp = (* generates return statement in java *)
  (*let output = (gen_expr exp) in*)
  sprintf "return %s;" (string_of_expr_t exp)*)*)
```

GraphQuil.ml

```
open Unix
open Printf

let java_compiler = "javac"

type action = Ast | SymbolTable | Sast | SAnalysis | Intermediate | Compile |
Java | Help

let usage (name:string) =
  "usage:\n" ^ name ^ "\n" ^
    "        -a source.gq                 (Print AST of source)\n" ^
    "        -sa source.gq                (Print SAST of source)\n" ^
    "        -t source.gq                 (Print Symbol Table of source)\n" ^
    "        -s source.gq                 (Run Semantic Analysis over source)\n"
^
    "        -i source.gq                 (Print intermediate representation of
source)\n" ^
    "        -j source.gq [target.java]   (Compile to Java executable)\n" ^
    "        -c source.gq                 (Compile source)\n"

let get_compiler_path (path:string) =
try
let i = String.rindex path '/' in
String.sub path 0 i
with _ -> "."
let _ =
  let action =
  if Array.length Sys.argv > 1 then
    (match Sys.argv.(1) with
        "-a" -> if Array.length Sys.argv == 3 then Ast else Help
      | "-sa" -> if Array.length Sys.argv == 3 then Sast else Help
      | "-t" -> if Array.length Sys.argv == 3 then SymbolTable else Help
      | "-s" -> if Array.length Sys.argv == 3 then SAnalysis else Help
      | "-i" -> if Array.length Sys.argv == 3 then Intermediate else Help
      | "-j" -> if Array.length Sys.argv == 3 then Java else Help
      | "-c" -> if Array.length Sys.argv == 3 then Compile else Help
      | _ -> Help)
  else Help in

  match action with
      Help -> print_endline (usage Sys.argv.(0))
    | (Ast | SymbolTable | Sast| SAnalysis | Intermediate | Compile | Java ) ->
      let input = open_in Sys.argv.(2) in
      let lexbuf = Lexing.from_channel input in
      let program = Parser.program Scanner.token lexbuf in
      match action with
          Ast -> let listing = Ast.string_of_prog program
                 in print_string listing
        | SymbolTable -> let env = SymbolTable.symbol_table_of_prog program in
                    print_string (SymbolTable.string_of_symbol_table env ^ "\n")
        | SAnalysis -> let env = SymbolTable.symbol_table_of_prog program in
                    let checked = Semantic_check.check_program program env
(SymbolTable.empty_tag_map) in
                    ignore checked; print_string "Passed Semantic Analysis.\n"
        | Sast -> let env = SymbolTable.symbol_table_of_prog program in
                    let checked = Semantic_check.check_program program env
(SymbolTable.empty_tag_map) in
                      let listing_t = Sast.string_of_prog_t checked in
                        print_string listing_t

        | Compile -> let env = SymbolTable.symbol_table_of_prog program in
                     let checked = Semantic_check.check_program program env
(SymbolTable.empty_tag_map) in
                     Javagen.write_code "graphQuil" checked; print_string
"compiled"
```

```
        | Java -> let env = SymbolTable.symbol_table_of_prog program in
                     let checked = Semantic_check.check_program program env
(SymbolTable.empty_tag_map) in
                     let execuatble_file_name = Javagen.write_code "graphQuil"
checked in
                    let dot_java_file = execuatble_file_name ^ ".java" in
                    execvp java_compiler [|"-g:none"; dot_java_file|] ;
        | Help -> print_endline (usage Sys.argv.(0)) (* impossible case *)
```

Makefile

```
#
# Author: Gemma Ragozzine
# Thank you to Edward's MicroC example, the Lorax example from Fall 2013, and
the wdjc example from Fall 2013
#

OBJS = ast.cmo sast.cmo symbolTable.cmo semantic_check.cmo parser.cmo
scanner.cmo javagen.cmo graphQuil.cmo
# OBJS = ast.cmo sast.cmo symbolTable.cmo semantic_check.cmo parser.cmo
scanner.cmo produceJava.cmo graphQuil.cmo

graphquil : $(OBJS)
	ocamlc -o graphquil -g unix.cma $(OBJS)

.PHONY : test
test : graphquil testall.sh
	./testall.sh

scanner.ml : scanner.mll
	ocamllex scanner.mll

parser.ml parser.mli : parser.mly
	ocamlyacc parser.mly

%.cmo : %.ml
	ocamlc -c -g $<

%.cmi : %.mli
	ocamlc -c -g $<

.PHONY : clean
clean :
	rm -rf graphquil parser.ml parser.mli scanner.ml testall.log \
	*.cmo *.cmi *.out *.diff *~  a.out.dSYM

# Generated by ocamldep *.ml *.mli
ast.cmo:
ast.cmx:
sast.cmo: ast.cmo
sast.cmx: ast.cmx
```

```
symbolTable.cmo: ast.cmo sast.cmo
symbolTable.cmx: ast.cmx sast.cmx
semantic_check.cmo: symbolTable.cmo
semantic_check.cmx: symbolTable.cmx
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmo
javagen.cmo: scanner.cmo parser.cmi ast.cmo sast.cmo symbolTable.cmo
semantic_check.cmo
javagen.cmx: scanner.cmx parser.cmx ast.cmx sast.cmx symbolTable.cmx
semantic_check.cmx
graphquil.cmo: scanner.cmo parser.cmi ast.cmo sast.cmo symbolTable.cmo
semantic_check.cmo javagen.cmo
graphquil.cmx: scanner.cmx parser.cmx ast.cmx sast.cmx symbolTable.cmx
semantic_check.cmx javagen.cmx
```

Testall.sh

```
#!/bin/sh

# Author: Gemma Ragozzine
# Thank you to Stephen Edward's MicroC example and Lorax Example from Fall 2013
#

graphquil="./graphquil"
java_output="java graphQuil"
java_file="graphQuil"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0
chmod +x java

Usage() {
    echo "Usage: testall.sh [options] [.gq files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
       echo "FAILED"
       error=1
    fi
    echo "  $1"
}

# Compare <outfile> <expected_file> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
       SignalError "$1 differs"
       echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
       SignalError "$1 failed on $*"
       return 1
    }
}


CheckParser() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.gq//'`
    reffile=`echo $1 | sed 's/.gq$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.a.out" &&
    Run "$graphquil" "-a" $1 ">" ${basename}.a.out &&
    Compare ${basename}.a.out ${reffile}.out ${basename}.a.diff

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}
```

```
CheckSemanticAnalysis() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.gq//'`
    reffile=`echo $1 | sed 's/.gq$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.s.out" &&
    Run "$graphquil" "-s" $1 ">" ${basename}.s.out &&
    Compare ${basename}.s.out ${reffile}.out ${basename}.s.diff

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}

#Right now, tests failures up to semantic check point
CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.gq//'`
    reffile=`echo $1 | sed 's/.gq$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    # old from graphquil - interpreter
    # generatedfiles="$generatedfiles ${basename}.i.out" &&
    # Run "$graphquil" "-i" "<" $1 ">" ${basename}.i.out &&
    # Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff

    generatedfiles="$generatedfiles ${basename}.s.out" &&
    {Run "$graphquil" "-s" $1 "2>" ${basename}.s.out} &&
    Compare ${basename}.s.out ${reffile}.out ${basename}.s.diff
    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.gq//'`
    reffile=`echo $1 | sed 's/.gq$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    #generatedfiles="$generatedfiles ${basename}.i.out" &&
    #Run "$graphquil" "-i" "<" $1 ">" ${basename}.i.out &&
    #Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff

    generatedfiles="$generatedfiles ${basename}.j.out" &&
    Run "$graphquil" "-c" $1 ">" ${basename}.j.out &&
    Compare ${basename}.j.out ${reffile}.out ${basename}.j.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
           rm -f $generatedfiles
       fi
       echo "OK"
       echo "###### SUCCESS" 1>&2
    else
       echo "###### FAILED" 1>&2
       globalerror=$error
    fi
}

TestRunningProgram() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
```

```
                             s/.gq//'`
    reffile=`echo $1 | sed 's/.gq$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    # echo -n "$basename..."
    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""
    tmpfiles=""

    # old from microc - interpreter
    # generatedfiles="$generatedfiles ${basename}.i.out" &&
    # Run "$lorax" "-i" "<" $1 ">" ${basename}.i.out &&
    # Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff

    generatedfiles="$generatedfiles ${basename}.j.out" &&
    tmpfiles="$tmpfiles " &&
    Run "$graphquil" "-j" $1 &&
    Run "$java_output" ">" ${basename}.j.out &&
    Compare ${basename}.j.out ${reffile}.out ${basename}.f.diff

    rm -f $tmpfiles

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
       k) # Keep intermediate files
           keep=1
           ;;
       h) # Help
           Usage
           ;;
    esac
done

shift `expr $OPTIND - 1`

if [ $# -ge 1 ]
```

```
then
    files=$@
else
    files="tests/test-*.gq"
fi

for file in $files
do
    case $file in
    *test-parser*)
        CheckParser $file 2>> $globallog
        ;;
    *test-sc*)
        CheckSemanticAnalysis $file 2>> $globallog
        ;;
    *test-full*)
        TestRunningProgram $file 2>> $globallog
        ;;
    *test-fail*)
        CheckFail $file 2>> $globallog
        ;;
	*test-*)
	    Check $file 2>> $globallog
	    ;;
	*)
	    echo "unknown file type $file"
	    globalerror=1
	    ;;
    esac
done

exit $globalerror
```

10.2 Appendix B: Test Suites

Tests to fail:

1. test-fail1.gq

```
/*
 * Author: Gemma Ragozzine
 * Tests failure of conditional statements
 */

int main () {

       int i;
       i = 1;
       while(i) {
              test();
       }
}
```

```
int test(){
	1 + 2;
}
```

2. test-fail2.gq

```
/*
 * Author: Gemma Ragozzine
 * Tests function return types
 */


int function()
{
	return true;
}
int main () {
	function();
}
```

3. test-fail3.gq

```
/*
 * Author: Gemma Ragozzine
 * Tests for scanner error
 */

int main () {

	int i;
	i = 1;
	i?2;
}
```

4. test-fail4.gq

```
/*
 * Author: Gemma Ragozzine
 * Tests wrong types
 */

int main () {
	String t;
	1 + t;
}
```

5. test-fail5.gq

```
/*
 * no main method
 * Author: Gemma
 */

void func1(){
       int a;
       int b;
}

int func2(int c, String k){
       return c;
}
```

6. test-fail6.gq

```
/*
 * Failure - no main method
 * Author: Gemma
 */

void func1(){
       int a;
       int b;
}

int func2(int c, String k){
       return c;
}
```

7. test-fail7.gq

```
/*
 * Author: Gemma Ragozzine
 * Test formal arguments of main method
 */

void main(int j, int k){
       String a;
}
```

8. test-fail8.gq

```
/* Author: Gemma Ragozzine
 * Tests return type of main
 */

String main(){
       int j;
       j = 1;
}
```

Full test:

1. test-full1.gq

```
void main()
{
  print(39 + 3);
}
```

Parser tests:

1. test-parser1.gq

```
void main()
{
print(4);
}
```

2. test-parser2.gq

```
int func1() {
 bool be;
 be = false;
}

int func2() {
print("hi");
}

int main()
{
func1();
func2();
}
```

3. test-parser3.gq

```
void main()
{
	int a;
	String b;
	while(true) {
		a = 1;
		b = "gemma";
		print(a);
		print(b);
	}
}
```

4. test-parser4.gq

```
/*
 *Test node creation
 */

void main(){
	int out;
	Edge b;

	b add ["height":6];
	out = b["height"];
}
```

5. test-parser5.gq

```
/*
 *Test basic operations
 */

void main(){
	int g;
	int out;
	out = 7;

	if(out == 7)  {
		int a;
		out = 8;
	}




	testfunc();
}

int testfunc(){
		int k;
		k = 4;

		k = g;
		return k;
}
```

Semantic check tests

1. test-sc1.gq

```
String carName;


void main(){
	carName = "Toyota";
	startCar(carName);
}

void startCar(String car_name){
	bool on;
	on = false;
	drive(car_name, on);
}

void drive(String car, bool on_off){
	while(on_off){

	}
}
```

2. test-sc2.gq

```
/*
 * Author: Gemma
 * Testing valid semantic analysis
 */

void main()
{
	bool testbool;
	int g;
	while(testbool)
	{
		for(g = 0; g < 4; g = g + 1)
		{

		}
	}
}
```

3. test-sc3.gq

```
/*
 * Author: Gemma
 * Testing valid semantic analysis
 */

void main()
{
	bool testbool;
	{
		if(testbool)
		{

		}
	}
}
```

4. test-sc4.gq

```
/*
 * Author: Gemma Ragozzine
 * Checks if/else loops and print arguments
 */
int foo(bool a, int b) {
  int i;
  if (a){
    return b;
   }
   else{
    for (i = 0 ; i < 5 ; i = i+1){
       b = b + 5;
       return b;
    }
  }
}

void main(){
       print(foo(true,45));
       print(foo(false,45));
}
```

5. test-sc5.gq

```
/*
 * Author: Gemma Ragozzine
 * Tests while loops and print
 */

void main()
{
  int g;
  g = 5;
  while (g > 0) {
    print(g);
    g = g - 1;
  }
  print(42);
}
```

6. test-sc6.gq

```
/*
 * Author: Gemma Ragozzine
 * Tests valid semantic analysis on operators
 */

void main(){
       21 + 64;
       'g' + 'a';
       true && false;

}
```

7. test-sc7.gq

```
/*
 * Author: Gemma Ragozzine
 * Testing valid semantic analysis
 */
```

```
int g; /* global scope */

void main(){
       int k;
       int g; /* scope of main function */

}
```