GRAPL: A GRAph Processing Language

COMS4115 (PLT) Final Project Report

December 22, 2010

1 Introduction

This report is the definitive reference on the motivation history, usage, and development of the **GRAPL** (**GRA**ph **P**rocessing **L**anguage), a simple user-friendly language designed to simplify the creation and navigation of directed graphs. We are confident that you will not find a more thoroughly researched compendium of GRAPL information anywhere.

1.1 White Paper

We reprint here—unedited and in its original format—the GRAPL white paper / project proposal from September 29, 2010. Note that the Language Reference Manual (Section 3) contains more up-to-date information on GRAPL's modern incarnation.

PLT Project Proposal: GRAPL (Graph Processing Language)

September 29, 2010

1. Introduction & Motivation

GRAPL (**GRA**ph **P**rocessing **L**anguage) is a user-friendly language designed to simplify the creation and navigation of directed graphs. Creating graphs in existing languages generally requires building node objects and maintaining arrays or lists of pointers to other nodes (along with weights); keeping track of this information can be cumbersome. In addition, the structure of the graph (in the general case where any node may be connected to any other node) takes a good deal of code to set up. GRAPL is intended to hide the complexity of graph navigation and to make creating graphs and adding nodes as simple as possible.

2. Language Features

GRAPL provides the following features:

Graph construction primitives. Many difficulties arise from trying to specify in a language a simple drawing. Having this in mind, GRAPL aims to make the construction of a graph as simple as drawing one. With GRAPL you can build a complex graph with as few as a pair of lines of code, using convenient operators such as >w> for directed edges and <w> for bidirectional edges with weights. The nodes and edges are automatically induced by the GRAPL compiler.

Easy graph manipulation. Built in control flow routines make graph algorithm programming as simple as writing a typical hello world program.

Four different data types. Graph manipulation shouldn't deal with various different data types. The only data types a GRAPL programmer should worry about are the basics: Node, Edge, Graph, and Number.

User defined functions. The user can build his own function to later reuse in his programs as many times as he likes.

3. Language Overview

3.1 Identifiers, Operators, and Data Types

3.1.1 **Identifiers.** Identifiers are a sequence of English characters (both upper and lower case), digits and underlines. The first token of an identifier should be a character. Other special symbols such as &, *, and ! are not allowed in identifiers.

3.1.2 Operators

3.1.2.1 Arithmetic Operators

"+", "-", "*", and "/" are used in numeric calculations. "<", ">", "<=", ">=", "==" are used arithmetically in number comparisons.

3.1.2.2 Special Operators

"Node1" + "<" + "Weight" + ">" + "Node2" are used in graph generations, meaning creation of two new nodes and a new bidirectional edge connecting them with weight shown in the middle.

"Node1" + ">" + "Weight" + ">" + "Node2" are used in graph generations, meaning creation of two new nodes and a new unidirectional edge from "Node1" to "Node2" with weight shown in the middle.

"List" + "+" + "Node" are used to mean adding a node to the end of a list.

3.1.3 Data Types / Objects

3.1.3.1 A number data type is a primitive declared using the keyword `Number`. This data type contains integer part and decimal fraction. *Example:* `Number num = 5.3.`

3.1.3.2 A string data type is a primitive declared using the keyword `String`.
Example: `String str = "Big Panda!"`

3.1.3.3 **Node** objects are created automatically in the creation of a new graph, without explicit declaration. They can also be declared individually. Nodes have a few built-in methods and/or public attributes, including:

- · `node.visit()` // marks node as visited
- · `node.isVisited()` // returns Boolean
- · `node.unvisit()` // removes visitor marking
- · `node.print()` // prints the name of the node
- · `node.id()` // returns a unique identifier for the node

3.1.3.4 Like nodes, **Edges** are also created automatically (implicitly) in the creation of a new graph, but may be declared explicitly. Edges have an edge weight, which defaults to zero if not specified.

3.1.3.5 A **List** data type is declared using the keyword `List`**.** This data type represents lists of nodes. *Example:* `List list = {Node node1, Node node2}.` Lists may have a few public attributes and/or methods, e.g:

- · `list.length()` // get the length of a list
- · `list.print()` // print out the contents of a list

3.1.3.6 A **Graph** data type is declared using the keyword `Graph`. This data type represents graphs composed of nodes and edges.

Example: *`Graph g = [A <2> B >3> C <4> D <5> B, C <1> E];  // A, B, C, D, E are nodes`*

This example a graph with nodes A, B, C, D, E. A is connected to B with one (weight = 2) bidirectional edge. B is connected to C with one (weight = 3) unidirectional edge from B to C. The other nodes and edges are created in the same mechanism. Graphs have a few built-in methods and/or public attributes, including:

- · `graph.clearVisited()` // clears Visited flag on all nodes in graph
- · `graph.print()` // prints some reasonably useful representation of the graph
- · `graph1.insert(graph2)` // inserts the argument into graph1. graph2 could be a single node or a graph structure. Nodes with the same names as those in the graph1 are considered to be the same objects.

3.2 Control Structures

3.2.1 **If-then-else.** This works exactly as the if-then-else structure in languages like C and JAVA.

3.2.2 **forEach iterator.** This special iterator at the heart of GRAPL vastly simplifies graph traversal. The syntax is as follows:

forEach { <optionally> visited | unvisited } node1 { from | to } node0 { <optionally> withEdge (operator Number, operator Number...) }

forEach loops through all nodes "node1" that are either parents or children of node0 (depending on the “from” or “to” keyword) and satisfy the conditions (visited, unvisited, or with a constraint on edge weight). "Operator" is one of { < , > , = , <= , >= }.

Example:

```
forEach unvisited child from startNode withEdge (>10)
{
      /* do something */
}
```

3.3 **User Functions.** Users are allowed to create their own functions and implement their algorithms using this language. Function declarations are in the C style.

3.4 **Recursion.** Obviously, GRAPL supports recursive function calls; most algorithms are expected to be recursive. No “rec” keyword is needed.

4. Sample Code

4.1. Initialize and Modify Graph

```
void main(){      // Add some edges to the graph
      graph g = [a >3> b >4> c <5> d];
      // Modify the graph later if we want add more things.
      g.insert(b>4>a);
      // We can store node information in list
      List ll = null;         ll + a;     // append node a        ll + b;
      // append node b        ll.print;   // output : { a, b}.

      // Build a empty list to store the results for dfs          List l =
null;       l = dfs(l,a); // output: a,b,c,d          l.print();        List
myPath = null;          myPath = path(path,a ,c); // output a, b, c
      myPath.print();     }
```

4.2 Depth First Search

```
list dfs (list l, node n) {
```

```
      n.visit();
      l + n; // appends current node n to the list
      forEach unvisited m from n { dfs (l, m); }      return l; }
```

4.3 Finding A Path To A Node

```
list path(list l, node n, node m){
      n.visit();
      l + n;
      if (n.id() == m.id())
            return l;         else
            forEach unvisited temp from n
            {
                  path(l, temp, m);
            }
}
```

4.4. Finding the node with most related nodes. Starting with node n_0, look for the node n such that 1) the distance between n_0 and n is less than 1800; and 2) n is the node in the graph (except n_0) that has the most related nodes (related nodes mean that n has at least one path to arrive at) within the distance of 100. This might be used, for example, to model the density of surrounding living areas in order to choose the best site for a new business.

```
void FindOptimalNode(Node startNode)
{
      Node nOpt;
      Number num = 0;
      forEach n from n0 withWeight (<1800)
      {
            Number num'=0;
            forEach n' from n withWeight (<100)
            {
                  num'= num' + 1;
            }
      }

      if (num' > num)
      {
            num = num';
            nOpt = n';
      }
      nOpt.print();
}
```

5.0 Future Directions and Add-ons

If time permits, we have a number of ideas for extending the language. These include:

1. 1. Including standard library functions like depth-first-search, BFS, Dijkstra, etc. (Some of these may be built in to the language).
2. 2. Extending the language to permit creation of text-based-adventure games. This would require storing additional information at each node (probably in a dictionary of some kind), and an additional notion of a User object that would navigate through the graph in an interpreted session.
3. 3. Building a simple GUI to display a created graph in a readable fashion.
4. 4. Building a GUI that allows the creation of nodes and edges graphically. (Yeah, right.)

6.0 Team Information

Team GRAPL (aka Mandarin Express) consists of:

- Di Wen (dw2464@columbia.edu)
- Yang Yi (yy2339@columbia.edu)
- Lili Chen (lc2737@columbia.edu)
- Andres Uribe (au2158@columbia.edu)
- Ryan Turner (rct2115@columbia.edu)

2 Language Tutorial

2.1 Using Primitive types

GRAPL has number as the primitive types, and allow arithmetic operators, plus(+), minus(-), multiple(*), divide(/) for the number.

Example 2.2 – Using number:

```
void main(){

    number n1;
    number n2;
    number n3;

   n1=4;
   n2=6;
    n3 = 3 * 4;
    print(n3);

    n3 = 3 / 4;
    print(n3);

    n3 = n1 * n2;
    print(n3);

    n3 = n2 / n1;
    print(n3);

    n3 = n2 + n1;
    print(n3);

    n3 = n2 - n1;
    print(n3);
}
```

Output:

12.0
0.75
24.0
1.5

10.0
2.0

2.2 Using Comments

GRAPL uses C-style, un-nested comments; that is, the characters `/*` introduce a comment, which terminates with the characters `*/`.

2.3 Using Node and List

GRAPL has node. A List is a collection of nodes. Each node has its own name. GRAPL is able to initialize a List with pre-defined nodes, or add new nodes to it using the concatenation operator ::. GRAPL can also tell the length of a list by calling the length function. Node has a property indicating if the node itself is visited or not. Just use visit() or unvisit() function.

Example 2.4– Using node and List:

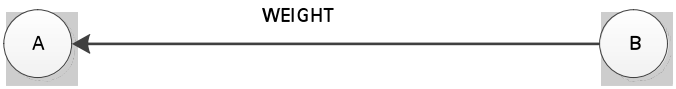
```
void main() {

    node n1;
    node n2;
    node n3;
    node n4;
    number nn;
    list l3;
    list l1;
    list l2;
    l2 = [n1,n2];
    l3 = [n2, n3, n4];
    nn = length(l3);
    l3 = n1 :: l3;
    n2 = head(l3);
    l2 = tail(l3);
    print(nn);
    print(l3);
}
```

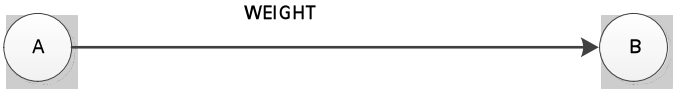
Output:
3.0
list: n3 n4 n1

2.4 Build a Graph

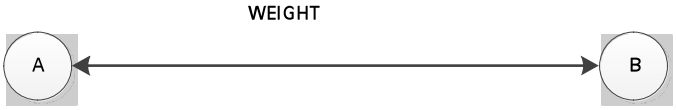
GRAPL's main feature now!! GRAPL provides a simple way to initialize a graph. With node A << weight Node B, Node A >> weight Node B , Node A <> weight Node B represents the following relationships between Node A and Node B respectively.



A << weight Node B



A >> weight Node B



A <> weight Node B

We will show the usage of graph initialization in later example combined with other statements.

2.5 Control Statement

GRAPL provides if-then-else and while to control the program flow. The following examples combined the usage of list with these two control statements.

Example 2.6.1– Using if-then-else:

```
void main()
{
node n1;
node n2;
node n3;
node n4;
number len;

list l1;
list l2;

l1 = [n1,n2,n3,n4];

visit n4;
len = length(l1);

if(len == 3) then
{
print(l1);

}
```

```
else
{

if (len > 3) then
{
print(len);
}

else {}
}

}
```

Output:
4.0

Example 2.6.2– Using while

```
void main() {

node n1;
node n2;
node n3;
node n4;
node n;
list l1;
l1 = [n1,n2,n3,n4];



while(length(l1) > 1)
{
      l1 = tail(l1);
      print(l1);
      print(head(l1));
}
}
```

Output:
list: n2 n3 n4
node n2: not visited.
list: n3 n4
node n3: not visited.
list: n4
node n4: not visited

2.6 Traverse a Graph

GRAPL provides a really simple way to traverse a graph with specified properties. The basic usage of the forEach function is:

foreach-statement:
`foreach` *node-control { statements }*

node-control:
qualifiers_opt id fromTo id withStmt

qualifiers_opt:
`visited`
`unvisited`

fromTo:
`from`
`to`

withStmt:
`with` *(predicate)*

predicate:
binary-operator expression

Example:

```
forEach visited leaf from root with (<= 1000)
{
/*do something*/
}
```

Example 2.7 – Using forEach to traverse a graph

```
node n1;
void main()
{

graph [n1 <<1 n2 <>2 n3, n4 <>2 n2, n5 >>3 n2];

forEach unvisited n from n2 with ( < 2 ) {

      print(n);
    visit n;

}

forEach unvisited n from n2 with ( <= 1 ) {
      print(n);
    unvisit n;

}
```

```
visit n5;
visit n3;
forEach visited n to n2 with ( > 2 ) {

    print(n);

}

forEach visited n to n2 with ( >= 3 ) {

    print(n);

}

}
```

Output:
node n1: not visited.
node n5: visited.
node n5: visited.

2.7 The Print function

In GRAPL we have the build-in print function to print help print different kinds of types, number, node, list, graph. Later example will show the usage of the print function.

2.8 More advanced algorithms

GRAPL enables people to build more advanced algorithm using recursion. Here are two simple examples.

Example 2.8.1 – a simple recursive function

```
void simpleRecursive(node finish)
{
  visit finish;
  forEach unvisited n to finish
  {
    print(n);
    visit n;
    simpleRecursive(n);
  }
}
void main()
{
  graph [a <> b, z <> b,  a >> c, c<>d, z<>e];
  simpleRecursive(b);
}
```

Output:
node a: not visited.

node z: not visited.
node e: not visited.

Example 2.8.2– a depth first search algorithm implementation using recursion

```
void main(){
 graph [a >>b >>c>>e>>f];
 print(dfs(a, f));
}

list dfs (node start, node finish) {
 list l;
 unvisitAll();
 return dfs_helper(start, finish,l);
}
list dfs_helper(node start, node finish, list l)
{
 visit start;
 if (start != finish) then
 {
  forEach unvisited n from start
  {
  n::l;
  if(n!=finish)then{
   dfs_helper(n, finish,l);
   }
   else
   { return l; }

  }}
 else
 {}
 return l;
}
```

Output:
list: b c e f

2.9 Standard Library

To facilitate users, GRAPL has a standard library, implemented simple algorithms such as the depth first search algorithm, un-Visit a certain sequence of node, etc. See section 3.7 of the LRM.

3 GRAPL Language Reference Manual

revised and updated 12/22/10

3.1 Introduction

GRAPL (**GRA**ph **P**rocessing **L**anguage) is a user-friendly language designed to simplify the creation and navigation of directed graphs. Creating graphs in existing languages generally

requires building node objects and maintaining arrays or lists of pointers to other nodes (along with weights); keeping track of this information can be cumbersome. In addition, the structure of the graph (in the general case where any node may be connected to any other node) takes a good deal of code to set up. GRAPL is intended to hide the complexity of graph navigation and to make creating graphs and adding nodes as simple as possible.

This manual describes the syntax and use of the GRAPL language. It is an unabashed plagiarism of the C Language Reference Manual (CLRM), published as Appendix A in Kernighan and Ritchie's "The C Programming Language." Where GRAPL and C share common features, we have sometimes reproduced the corresponding text from the CLRM verbatim rather than attempt to paraphrase an identical idea.

3.2 Lexical Conventions

There are six classes of tokens: identifiers, keywords, constants, strings, operators, and other separators. As in C, whitespace characters are ignored except insofar as they serve to delineate other tokens in the input stream. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

3.2.1 Comments

GRAPL uses C-style, un-nested comments; that is, the characters `/*` introduce a comment, which terminates with the characters `*/`.

3.2.2 Identifiers

A GRAPL identifier must meet the same requirements as a C identifier; that is, it consists of a sequence of letters, digits, and underscore characters, where the first character must be either a letter or an underscore. Identifiers are case-sensitive.

3.2.3 Keywords

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

```
boolean          length          unvisit
else             list            unvisited
forEach          node            visit
from             number          visited
head             print
if               return          with
isVisited        tail
graph            to
```

3.2.4 Literals

The only literals in GRAPL are of type `number` and `boolean.` A `number` is analogous to a `double` representation in C.

3.3 Expressions

Expressions, along with operators, are essential elements in the formation of code in the GRAPL language. Here we try to specify all of the expressions with operators.

The precedence of all the operators is the same as the order of the subsections here, that is to say, all operators in sections 3.1 – 3.5 have higher precedence in comparison with operators in section 3.6. Besides, all operators in the same subsection have the same precedence.

We don't assume either left or right associativity for the expressions. For evaluation of expressions, the order is also set to be undefined.

For handling arithmetic problems, we define that overflow of numbers will be discarded. In addition, problems like "divided by zero" will cause exception and be handled in library functions.

The following table summarizes the rules:

Tokens	**Operators**	**Associativity**
Identifiers, parenthesized expression	Primary	
`( )`	Function calls	Left
`head`	List access (head)	Left
`tail`	List access (tail)	Left
`visit unvisit isvisited`	Node Visitation	Right
`!`	Negation	Right
`* /`	Multiplicative	Left
`+ -`	Additive	Left
`< <= > >=`	Relational	Left
`== !=`	Equality	Left
`::`	List concatenation	Right
`=`	Assignment	Right

3.3.1 Primary Expressions

Primary expressions are identifiers, constants, or expressions in parentheses.

primary-expression:
identifier
literal
(expression)

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

A literal constant is a primary expression.

An expression inside parentheses is a primary expression.

3.3.2 Function calls

Expressions involving function calls associate from left to right. The following is the syntax used:

fcall-expression:
function-identifier(argument-list-opt)

argument-list-opt:
nothing
argument-list

argument-list:
expression
argument-list , expression

The function call will return the type specified by the function declaration. In order to be correctly evaluated the function has to be in scope.

3.3.3 Node operators

There are three unary node operators, `visit`, `unvisit`, `isVisited`. The are all right-associative. The first two operators mark a node as having been visited or unvisited, respectively, while the last returns a Boolean value indicating the visitation status of the node.

visit-expression:
`visit` *node-identifier*

unvisit-expression
`unvisit` *node-identifier*

isVisited-expression:
`isVisited` *node-identifier*

3.3.4 Negation Operator

The negation operator returns the opposite boolean value of its argument. The right hand side must evaluate to a Boolean type.

negation-expression:
! expression

3.3.5 Multiplicative Operators

The multiplicative operators * and / group left-to-right.

multiplicative-expression:
*expression * expression*
expression / expression

The operands of * and / must have number type.

Operations are carried on the operands and the result value is returned.

The binary * operator denotes multiplication of the left and right operands.

The binary / operator denotes division. The left operand is divided by the right operand. If the right operand is 0, the value is undefined and an exception will be caught.

3.3.6 Additive Operator

The additive operators + and - group left-to-right.

additive-expression:
expression + expression
expression - expression

The operands of + and – should have number type.

Operations are carried on the operands and the result value is returned.

The binary + operator denotes adding the two operands and getting the sum.

The binary – operator denotes subtracting the right operand from the left operand and getting the difference between the two.

3.3.7 Relational Operator

The relational operators group left-to-right.

relational-expression:
expression < expression
expression > expression
expression <= expression
expression >= expression

The operands must have `number` type here. The return value also has type `number`. For all of the operators < (less than), > (greater than), <= (less than or equal to), >= (greater

than or equal to), the return value is set to be 1 if the specified relation is true, and it's set to be 0 if the specified relation is false.

3.3.8 Equality Operator

The equality operators group left-to-right.

equality-expression:
expression == expression
expression != expression

The operands must have `number` type here. The return value also has type `number`. For the operators == (equal to) and != (not equal to), the return value is set to 1 if the specified relation is true, and it's set to 0 if the specified relation is false.

In addition, equality operators have lower precedence than relational operators.

3.3.9 Logical AND Operator

The logical AND operators group left-to-right.

logical-and-expression:
expression && expression

The operands and the result should have `number` type.

The evaluation of the expressions is also left-to-right. The left operand is firstly evaluated. If its value is 0, then the result value is set to 0. If its value is 1, then the right operand is evaluated. If the value of the right operand is also 1, then the result value is set to 1. If the value of the right operand is 0, then the result value is set to 0.

3.3.10 Logical OR Operator

The logical OR operators group left-to-right.

logical-or-expression:
expression || expression

The operands and the result should have `number` type.

The evaluation of the expressions is also left-to-right. The left operand is firstly evaluated. If its value is 1, then the result value is set to 1. If its value is 0, then the right operand is evaluated. If the value of the right operand is also 0, then the result value is set to 0. If the value of the right operand is 1, then the result value is set to 1.

3.3.11 Assignment Expressions

The assignment operators group right-to-left.

assignment-expression:
identifier = expression

Assignments can be made to identifiers of types `list`, `number`, `boolean`, or `node`. The expression on the right-hand side must evaluate to the same type as the identifier on the left. Besides, the value of the assignment expression is the value of the left operand after the assignment process.

3.3.12 List Concatenation Operator

Additional nodes may be concatenated onto the front of an existing list using the :: operator.

list-concatenation-expression:
node-identifier :: list-identifier

Example:

```
list l = a :: l; // a is a node; l is a list
```

3.3.13 List Access Operators

The first node in a list can be accessed with the `head` operator. The expression returns a node type.

head-expression:
`head` *list-identifier*

Example:

```
node h = head l; // h is a node; l is a list
```

The tail of a list can be accessed with the tail operator. The expression returns a list type.

tail-expression:
`tail` *list-identifier*

Example:

```
list t = tail l; // t and l are lists
```

3.4 Declarations

A variable should always be declared first before it can be used. Declarations have the form:

declaration:

type-specifier declarator ;

type-specifier:
`node`
`graph`
`list`
`number`
`boolean`

A variable name cannot be declared twice. It is also illegal to declare multiple variables in the same line.

The declaration contains exactly one type specifier and one declarator. Each declarator contains exactly one identifier. Each declarator is taken to be an assertion that when a construction of the same form as the declarator appears in an expression, it yields an object of the indicated type.

3.4.1 Node

Node objects are created automatically in the creation of a new graph, without explicit declaration (see 4.2). They can also be declared individually. A node identifier is implicitly a reference to a node, and multiple references may point to the same node.

Node-declaration:
`node` *identifier*
`node` *identifier = node-identifier*

Example:
```
node A; // declare node individually
```

Nodes can be initialized in the same line in which they are assigned; for instance, by setting the identifier to point to the same node as another identifier.

Example:
```
node x = lastNode; // lastNode is already in existence
```

3.4.2 Graph

In our compiler, a graph declaration is actually an initialization of the graph at the same time. Each program has one or zero graphs; because of this, the type-specifier `graph` is not followed by an identifier in the declaration/initialization. To define a `graph`, one or more nodes must be simultaneously defined, as follows. If more than one `node` is defined, then edge definitions are required between each sequential `node` in the declaration.

A graph can be declared as follows:
graph-declaration :
graph [constructor-declarations];

constructor-declarations:
constructor-declaration
constructor-declaration, constructor-declarations

constructor-declaration:
node-id
node-id edge-definitions

edge-definitions:
edge-definition
edge-definitions edge-definition

edge-definition:
>> *node-identifier*
<< *node-identifier*
<> *node-identifier*
>> *literal node-identifier*
<< *literal node-identifier*
<>*literal node-identifier*

Example:

```
graph [A <<3 B];
graph [A <>2 B >>3 C <>4 D <>5 B, C <>1 E];
// A, B, C, D, E are nodes
```

3.4.3 List

List are like arrays. They are assemblies of Nodes. Lists are declared with the keyword `list`:

List-declaration:
`list` *identifier*
`list` *identifier* = *[list]*

list:
<nothing>
node-identifiers

node-identifiers:
node-identifier
node-identifiers, node-identifier

Example:

```
list l = [A, B, C, D];
```

3.5 Statements

In GRAPL, statements are of five different types:

- Expression
- Compound
- Conditional
- Iteration
- Return

3.5.1 Expression

An expression statement has the form:

Expression-statement:
expression ;

It is important to note that the expression is not optional, so an instruction consisting of a single semicolon is not considered correct.

3.5.2 Compound Statements

Compound statements (blocks) are used to group a set of different type of one or more statements. They follow the form:

compound-statement:
{ statement-list }

statement-list
statement
statement statement-list

Each of the statements in this compound-statement will be executed sequentially starting from left to right.

3.5.3 Conditional Statements

Selection statements are used to select which block of statements to execute based on the evaluation of a controlling expression. They follow the following form:

conditional-statement:
if (expression) then { statements } else { statements }

The `else` clause in the if-then-else statement is not optional and cannot be omitted; however, an empty statement may follow the `else`.

Example1:

```
if (node1 == node2) then
{
print(node1);
}
else { }
```

Example2:

```
if (length(list1) < length(list2)) then
     print(list2);
else
{
node1 :: list1;
print(list1);
}
```

3.5.4 Iteration

GRAPL has a single iteration statement to iterate through the nodes of a given graph: the `foreach` statement. This statement allows a user to specify an instruction set to execute for each of the nodes on the other side of an edge with a weight that satisfies certain conditions (the predicate) inside a graph. The general form of the `foreach` loop is:

iteration-statement:
`foreach` *node-control { statements }*

node-control:
qualifiers_opt id fromTo id withStmt

qualifiers_opt:
`visited`
`unvisited`

fromTo:
`from`
`to`

withStmt:
`with` *(predicate)*

predicate:
binary-operator expression

Example:

```
forEach visited leaf from root with (<= 1000)
{
/*do something*/
}
```

3.5.5 Return Statement

The return statement returns a function's value to its caller by means of one of the following forms:

return-statement:

`return ;`

`return` *expression ;*

The expression must evaluate to the correct return type for the function.

3.6 Scope

The scope rules in GRAPL follow these guidelines. An identifier is available after it has been declared. Additionally, an identifier that was used in a graph initialization need not be initialized. Every identifier is available from the point where it was used in the graph initialization until the end of the statement group enclosing it. Subsequent nested blocks have access to identifiers defined by any parent block. A variable with the same name as in a parent block cannot be declared.

A function identifier is available at any point after the left bracket of the function definition starts. After a function is declared, the identifier is locked and cannot be used to define a variable.

3.6.1 Built-in Functions

GRAPL a number of built-in functions, as follows:

3.6.2 `number length(list l)` **:** returns the length of a list

3.6.3 `void print(node n)`**:** prints the name and status (visited, unvisted) of a node

3.6.4 `void print(list l)`**:** prints the contents of a list

3.6.5 `void print(number n)`**:** prints the value of a literal

3.6.6 `node head(list l)`**:** returns the node at the head of a list

3.6.7 `list tail(list l)`**:** returns the tail of a list. It will fail if called on a list of one or zero elements.

3.6.8 `boolean isVisited(node n)`**:** returns true if the node is marked as visited; false otherwise

3.6.9 `void unvisitAll(n)`**:** a useful feature for resetting all markings in the graph to `unvisited`

3.7 Standard Library Functions

In addition to the built-in functions, the GRAPL compiler automatically imports a GRAPL Standard Library with a number of useful functions. These functions are written in GRAPL and must be pre-compiled. The GRAPL user can add additional functions to this library with only a few trivial changes to the compiler.

The Standard Library functions include:

3.7.1 `number numChildren(node n)`**:** returns the number of (immediate) children of a node

3.7.2 `number numAncestors(node n)`**:** returns a count of all ancestors accessible from a node

3.7.3 `list dfs(node start, node end)`**:** returns a list of nodes in the path from `start` to `end`

3.7.4 `list reverse(list old_list)`**:** returns a reversed list

4 Project Plan

Our long-range plan to implement the whole project was comprised of three stages:
1. Parser/Scanner development, with output routed to grapl-printer (intermediate product)
2. Compiler and java-printer development
3. Testing and debugging
4. Upgrades, refinements, standard library functions

4.1 Process

The process we used for project is that we meet every Thursday for at least two hours. In every meeting, we discuss about the solutions for problems, assign tasks for every team member and ask for suggestions from our team TA. If anyone has questions about his or her part, we will focus on it and try to solve it effectively.

For proposal and reference manual parts, we have discussions about ideas and then every team member complete several parts. After that, we mainly work on how to implement our language – GRAPL. We always get together in CS lounge or TA room, working on each part. Over the course of the project, we derived a number of development best-practices, including:

- Informing every team member by emails when there are any important changes in the code
- Assigning each part of the documentation to a different member and integrating the result.
- Sharing word processing templates to that styles and numbering are consistent when different chapters are integrated
- Always updating to the latest version on SVN before beginning work on the project.

4.2 Programming style guide

We also set some conventions for our programming:

1. Providing relevant descriptive comments before every function declaration so that every team member can understand it quickly.
Example:

```
(* Returns a function call to create a new edge in the
graph *)
let jast_of_edge_def edge_def = match edge_def with
     | (s1, Ast.Redge(w), s2) ->
                Jast.Expr(Jast.Call(Jast.Id("graph"),
"addEdge", [Jast.StringLit(s1); Jast.StringLit(s2);
Jast.Literal(w)]))
     | (s1, Ast.Ledge(w), s2) ->
                Jast.Expr(Jast.Call(Jast.Id("graph"),
"addEdge", [Jast.StringLit(s2); Jast.StringLit(s1);
Jast.Literal(w)]))
     | (s1, Ast.Bedge(w), s2) ->

     Jast.Expr(Jast.Call(Jast.Id("graph"),"addBEdge",
[Jast.StringLit(s1); Jast.StringLit(s2); Jast.Literal(w)]))
     | (s1, Ast.NoEdge(w), s2) -> Jast.Nostmt
```

2. For every variable name, function name and code name, we try to make them easy to understand and self–explanatory. For example, method names in Java use camel-case; in O'Caml, underscores (ironically, perhaps).

Example:

```
addEdge();
getNode();
```

4.3 Project timeline

What language do we want to create?	Sep 13th – Sep 20th
Project Proposal	Sep 29th
Language Reference Manual	Nov 3rd
Scanner works, begin work on parser	Nov 10th
Parser and Ast	Nov 22nd
Java backend files	Dec 1st - Dec 21st
GRAPL pretty printer	Dec 7th
Java AST	Dec 8 th - Dec 19th
Compiler	Dec 9th- Dec 21st
Java pretty printer	Dec 9th- Dec 21st
Makefile	Dec 18th
Test	Dec 6th – Dec 22nd
Standard library functions and integration	Dec 20th – Dec 22nd
Final Report	Dec 22nd

4.4 Roles and Responsibilities of each team member

Member	Roles and Responsibilities
Ryan Turner	parser, compiler, team leader
Andres Uribe	parser, compiler, standard library
Yi Yang	Java backend, standard library
Lili Chen	scanner, grapl- and java-printers, testing
Di Wen	parser, testing suite

4.5 Software development environment used

1. Environment:
   Operating System: We do our projects on windows, MAC and Linux.

2. Language:
   Ocamllex is used for scanner; Ocamlyacc is used for parser and rest compiler codes are implemented In Ocaml.

   Java is used to build the java backend classes.

3. Tools:
   IDE: By initializing OCaml plug – in, we use IDE eclipse to write OCmal code.

   SVN: SVN is really helpful for team project since we can share the code together. Every team member can check latest code, revise, debug and commit it.

   Command-line with Makefile: Compile Ocaml

   Netbeans: Coding for Java

5 Architectural Design

The GRAPL compiler architecture was designed to be a translator that converts a grapl program file (extension `.gpl`) to a Java source code file. The output is then compiled and run using the Java compiler and Java virtual machine respectively. The architecture was divided into different components which are described in the following sections.

5.1 Components

The main components in GRAPL are:

- scanner (`scanner.mll`)
- parser (`parser.mly`)
- grapl pretty printer (`gpp.ml`)
- translator (`translate.ml`) (sometimes referred to as “compiler”)

- Java pretty printer (`jpp.ml`)
- Java back-end
- grapl compiler (`graplc.ml`).

Figure 1 illustrates the process of compiling a GRAPL program.

5.2 Interfaces

The interfaces created were `ast.mli`, `jast.mli`, `Graph.java`, and `GraplLib.java`.

The `ast.mli` interface was offered by the Parser and consumed by the Translator and the Gpp components. It gives a description of the types of the elements inside a GRAPL abstract syntax tree.

The `jast.mli` interface was offered by the Translator and was consumed by the Jpp component. It gives a description of the types of the elements inside a stripped down version of a Java abstract syntax tree.

The `Graph.java` was offered by the Graph Java class in the Java back-end and consumed by the Translator. The `GraplLib.java` was offered by the GraplLib Java class in the Java back-end and consumed by the Converter. Both of these classes listed the available method calls into which a GRAPL AST type could be converted.

5.3 Implementation

The following table shows what files are included in each component and who implemented them.

Component	File name	LC	RT	AU	DW	YY
scanner	scanner.mll	X				
parser	parser.mly		X	X	X	
Gpp	gpp.ml	X				
translator	compile.ml		X	X		
Jpp	jpp.ml	X				
back-end	java/lib/*.java					X
testing	test/*.gpl			X	X	
compiler			X			
Interface	ast.mli	X	X			
	jast.mli		X			
	Graph.java			X		X
	GraplLib.java			X		X

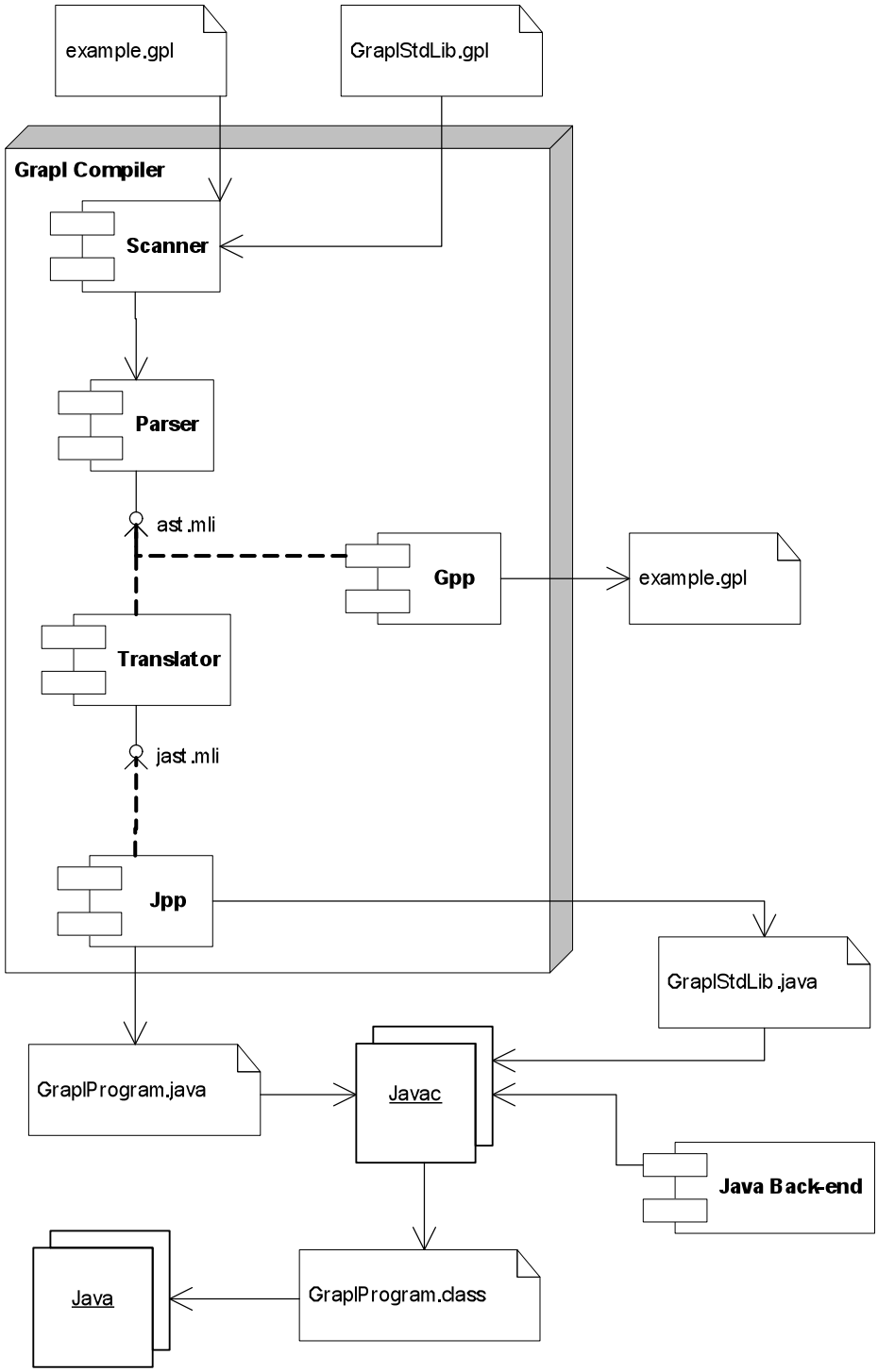


Figure 1: an overview of the GRAPL compiler architecture.

6 Test Plan

In testing, we mainly did two kinds of tests: functional testing and semantic testing.

For functional testing, we write a small source code, run it, and check whether the GRAPL code is accepted (by printing it using GRAPL pretty printer) and translated into JAVA (by printing it using JAVA pretty printer). In this procedure, parsing errors and functional errors are caught.

In semantic testing, we write small segments of erroneous source code which include some semantic errors, run it, and check whether the exceptions are caught.

Every member in the group contributes to testing. Andres and Di did the semantic checking.

For functional testing, we keep testing our functions in the process of development. Di wrote some test cases for different basic components of GRAPL, Ryan and Andres wrote some functions based on the basic components for testing, Yi and Lili made the list of the examples which were shown to Professor Edwards in presentation.

6.1 Representative Source Language Programs & The Target Language Programs

6.1.1 helloWorld.gpl

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

list helloWorld()
{
  node Hello;
  node World;
  list l;
  l = [Hello, World];
     return l;
}
```

6.1.2 helloWorld.java

```
import lib.*;


public class GraplProgram
{
public static GraplLib library;
public static Graph graph;

public static List helloWorld()
{

Node g_Hello = new Node("Hello");
Node g_World = new Node("World");
List g_l = new List();
g_l = library.buildList(g_Hello,g_World);
return g_l;
}


public static void main(String[] args)
{

library = new GraplLib();
graph = library.createEmptyGraph();
GraplStdLib.initialize(graph,library);
library.print(helloWorld());
}
}
```

6.1.3 forEach.gpl

```
node n1;
void main()
{

node n2;
node n3;
node n4;
node n5;
number x;

x=0;
```

```
graph [n1 <<1 n2 <>2 n3, n4 <>2 n2, n5 >>3 n2];
forEach unvisited n from n2
{
     x = x + 1;
     print(x);
}

x=0;

forEach unvisited n to n2
{
     x = x + 1;
     print(x);
}


forEach unvisited n from n2 with ( < 2 ) {

    visit n;

}

forEach unvisited n from n2 with ( <= 1 ) {

    unvisit n;

}

visit n5;
visit n3;
forEach visited n to n2 with ( > 2 ) {

    print(n);

}

forEach visited n to n2 with ( >= 3 ) {

    print(n);

}

}
```

6.1.4 forEach.java

```
import lib.*;
```

```
public class GraplProgram
{
public static Node g_n1;
public static GraplLib library;
public static Graph graph;

public static void main(String[] args)
{

g_n1 = new Node("g_n1");
library = new GraplLib();
graph = library.createEmptyGraph();
Node g_n2 = new Node("n2");
Node g_n3 = new Node("n3");
Node g_n4 = new Node("n4");
Node g_n5 = new Node("n5");
double g_x;
GraplStdLib.initialize(graph,library);
g_x = 0.;
graph.addEdge("n2","n1",1.);
graph.addBEdge("n2","n3",2.);
graph.addBEdge("n4","n2",2.);
graph.addEdge("n5","n2",3.);
for (Node g_n :
graph.getNodesFromWith(g_n2,null,"unvisited","",1.))
{
{
g_x = g_x + 1.;
library.print(g_x);
}
}

g_x = 0.;
for (Node g_n :
graph.getNodesToWith(g_n2,null,"unvisited","",1.))
{
{
g_x = g_x + 1.;
library.print(g_x);
}
}

for (Node g_n :
graph.getNodesFromWith(g_n2,null,"unvisited","<",2.))
{
{
g_n.visit();
```

```
}
}

for (Node g_n :
graph.getNodesFromWith(g_n2,null,"unvisited","<=",1.))
{
{
g_n.unvisit();
}
}

g_n5.visit();
g_n3.visit();
for (Node g_n :
graph.getNodesToWith(g_n2,null,"visited",">",2.))
{
{
library.print(g_n);
}
}

for (Node g_n :
graph.getNodesToWith(g_n2,null,"visited",">=",3.))
{
{
library.print(g_n);
}
}

}
}
```

6.1.5 dfsTest.gpl

```
void main(){
     graph [a <> b <> c <> a <>d <> e <> f <> d <> c <>f <>
b];
     print(dfs(a, f));
}
list dfs (node start, node finish) {
     unvisitAll();
     return dfs_helper(start, finish);
}
list dfs_helper(node start, node finish){
     list l;
     visit start;
```

```
        if (start == finish) then {
            start::l;
        }
        else {
            forEach unvisited n from start{
                l = dfs(n, finish);
                if( length( l ) != 0 ) then{
                    return l;
                }
                else {}
            }
        }
        return l;
    }
```

6.1.6 dfsTest.java

```
    import lib.*;


    public class GraplProgram
    {
    public static GraplLib library;
    public static Graph graph;

    public static List dfs_helper(Node g_start,Node g_finish)
    {

    List g_l = new List();
    g_start.visit();
    if (g_start == g_finish)
    {
    {
    g_l.addNew(g_start);
    }
    }
    else
    {
    {
    for (Node g_n :
    graph.getNodesFromWith(g_start,null,"unvisited","",1.))
    {
    {
    g_l = dfs(g_n,g_finish);
    if (g_l.size() != 0.)
    {
```

```
{
return g_l;
}
}
else
{
{
}
}
}
}

}
}
return g_l;
}


public static List dfs(Node g_start,Node g_finish)
{

graph.unvisitAll();
return dfs_helper(g_start,g_finish);
}


public static void main(String[] args)
{

library = new GraplLib();
graph = library.createEmptyGraph();
GraplStdLib.initialize(graph,library);
graph.addBEdge("a","b",1.);
graph.addBEdge("b","c",1.);
graph.addBEdge("c","a",1.);
graph.addBEdge("a","d",1.);
graph.addBEdge("d","e",1.);
graph.addBEdge("e","f",1.);
graph.addBEdge("f","d",1.);
graph.addBEdge("d","c",1.);
graph.addBEdge("c","f",1.);
graph.addBEdge("f","b",1.);
library.print(dfs(graph.getNode("a"),graph.getNode("f")));
}
}
```

6.2 Functional Test Cases

We write a lot of test cases to test the functions of different basic components of GRAPL. In order to avoid interference from unwanted independent variables, we try to make the test cases relatively small and progress from testing the simplest files to testing the complicated functions step by step.

6.2.1 simplestFile.gpl

```
node main(){
}
```

This is to test the main() function.

6.2.2 number.gpl

```
void main(){

    number n1;
    number n2;
    n1 = 1;
    n2 = 2;

    print(n1);
    print(n2);

}
```

This is to test the numbers.

6.2.3 AddandSub.gpl

```
void main(){

    number n1;
    number n2;
    number n3;

    n1 = 2;
    n2 = 3;

    n3 = 1 + 2;
    print(n3);

    n3 = 1 - 2;
    print(n3);

    n3 = n1 + n2;
```

```
    print(n3);

    n3 = n2 - n1;
    print(n3);

}
```

This is to test Addition and Subtraction.

6.2.4 MultandDiv.gpl

```
void main(){

    number n1;
    number n2;
    number n3;

    n1 = 2;
    n2 = 3;

    n3 = 3 * 4;
    print(n3);

    n3 = 3 / 4;
    print(n3);

    n3 = n1 * n2;
    print(n3);

    n3 = n2 / n1;
    print(n3);

}
```

This is to test Multiplication and Division.

6.2.5 boolean.gpl

```
void main() {

    number n1;
    number n2;
    boolean b;

    n1 = 1;
    n2 = 2;
```

```
        b = n1 < n2;

        b = n1 > n2;

        b = n1 <= n2;

        b = n1 >= n2;

        b = n1 == n2;

        b = n1 != n2;

    }
```

This is to test Boolean.

6.2.6 logicalOp.gpl

```
    void main() {

        number n1;
        number n2;
        boolean b1;
        boolean b2;
        boolean b;

        n1 = 1;
        n2 = 2;
        b1 = true;
        b2 = false;

        b = b1 == b2;

        b = b1 != b2;

        b = ! b1;

        b = ! (n1 < n2);

        b = b1 && b2;

        b = b1 || b2;

        b = (n1 < n2) && ( n1 > n2 );

        b = (n1 < n2) || ( n1 > n2 );
```

```
}
```

This is to test the logical operators: !, && and ||.

6.2.7 node.gpl

```
node global;
void main() {

  node n1;
  node n2;

  print(n1);
  print(n2);

  n2 = n1;
  visit n1;
  visit n2;
  unvisit n1;
  unvisit n2;

}
```

This is to test node, and the operations (visit, unvisit) on node.

6.2.8 list.gpl

```
void main() {

    node n1;
    node n2;
    node n3;
    node n4;
    list l3;
    number nn;
    list l1;
    list l2;
    n1 = n3;
    l2 = [n1,n2];
    l3 = [n2, n3, n4];
    nn = length(l3);
    n1 :: l3;
    n2 = head(l3);
    l2 = tail(l3);
}
```

This is to test list, and the operations (head, tail, length) on list.

6.2.9 function.gpl

```
node n1;
number testFunction (node n){
    node n2;
    list l;
     l = [n];
    l = n2 :: l;
    if(length(l) >= 2) then
          {
       print(l);
    }
     else
     {
        print(n);
    }
    /*return length(l);*/
}

void main(){

    number num;
    num = testFunction(n1);

}
```

This is to test calling another function in the main function.

6.2.10 graph.gpl

```
void helloGraph()
{
     node n1;
     node n2;
     graph [n1 >>1 n2];
}

void main(){

    helloGraph();

}
```

This is to test construction of a graph.

6.2.11 ifthenelse.gpl

```
void main()
{
node n1;
node n2;
node n3;
node n4;
number len;

list l1;
list l2;

l1 = [n1,n2,n3,n4];

visit n4;
len = length(l1);

if(len == 3) then
{
print(l1);

}

else
{

if (len > 3) then
{
print(len);
}

else {}
}

}
```

This is to test the if-then-else data flow.

6.2.12 while.gpl

```
void main() {
```

```
node n1;
node n2;
node n3;
node n4;
node n;
list l1;
l1 = [n1,n2,n3,n4];



while(length(l1) > 1)
{
     l1 = tail(l1);
     print(l1);
     print(head(l1));
}
}
```

This is to test the while loop.

6.2.13 forEach.gpl

```
node n1;
void main()
{

node n2;
node n3;
node n4;
node n5;
number x;

x=0;

graph [n1 <<1 n2 <>2 n3, n4 <>2 n2, n5 >>3 n2];
forEach unvisited n from n2
{
     x = x + 1;
     print(x);
}

x=0;

forEach unvisited n to n2
{
     x = x + 1;
     print(x);
```

```
}


forEach unvisited n from n2 with ( < 2 ) {

    visit n;

}

forEach unvisited n from n2 with ( <= 1 ) {

    unvisit n;

}

visit n5;
visit n3;
forEach visited n to n2 with ( > 2 ) {

    print(n);

}

forEach visited n to n2 with ( >= 3 ) {

    print(n);

}

}
```

This is to test the forEach loop.

6.2.14 Other functions

Now that all the data structures and flows are tested, we also write many programs based on these data types, functions and flows. We recursively test the cases, find problems and record problems, then correct them. Below is an example of tables showing the result of tests.

	-a	-c
AddandSub.gpl	Good	Can't Run
forEach.gpl	Nothing inside the forEach loop is shown	Can't Run
function.gpl	Nothing inside the if { } else { } is shown	Fatal error: exception Compile.AssignmentException ("Type error: 'l' cannot be as signed to [n]'.")
graph.gpl	Good	Good
helloWorld.gpl	Good	This function has nothing to do with graphs, but graph library and graph declarations are still called
ifthenelse.gpl	Nothing inside the if { } else { } is shown	Fatal error: exception Compile.AssignmentException ("Type error: 'node4' cannot be assigned to l1(tail)'.")
list.gpl	Good	Fatal error: exception Compile.AssignmentException ("Type error: 'nn' cannot be assigned to length(l3)'.")
logicalOp.gpl	Good	Can't Run
logicalOp2.gpl	Parse_Error	Parse_Error
MultandDiv.gpl	Good	Can't Run
node.gpl	Good	This function has nothing to do with lists and graphs, but lists and graphs are still included
number.gpl	Good	This function has nothing to do with graphs, but graph library and graph declarations are still called
simpleForEach.gpl	Nothing inside the forEach loop is shown	This function has nothing to do with lists, but lists are still included
simpleRecursive.gpl	Nothing inside the forEach loop is shown	This function has nothing to do with lists, but lists are still included
simplestFile.gpl	Good	This function has nothing to do with lists and graphs, but lists and graphs are still included
unaryBinary.gpl	Parse_Error	Parse_Error

6.3 Semantic Test Cases

We also write some test cases for semantic checking. These programs are "wrong" and will cause corresponding exceptions defined in compile.ml.

6.3.1 Return Value

```
void main(){

  number n;
  n = foo();

}
number foo(){

  node a;
  return a;

}
```

Here, a node is returned while the wanted return value is a number. We also write other test cases, such as return number while wanting boolean, return number while wanting node, return number while wanting list, etc.

6.3.2 If

```
void main(){
  if(1){
  }
}
```

Here, the predicate of if is an number, while it should be boolean. Similar tests, for instance, putting a node in predicate, are also used.

6.3.3 While

```
void main(){

  while(3){
  }
}
```

Similarly, the predicate of while-loop should be of boolean value. We use number instead to do semantic checking.

6.3.4 forEach

In forEach, the structure should be forEach + visited/unvisited + <node> + from/to + <node> + with + (==/!=/</>/<=/>= + <number>)

We write following test cases to check different tokens:

6.3.4.1 Check visited/unvisited

```
void main(){
  node n1;
  forEach 1 n from n1{
```

```
  }

}
```

6.3.4.2 Check <node>

```
void main(){

  node n1;

  forEach visited 3 from n1{
  }

}
```

6.3.4.3 Check from/to

```
void main(){

  node n1;

  forEach visited n 2 n1{
  }

}
```

6.3.4.4 Check <number>

```
void main(){

  node n1;
  node n2;

  graph [n1 <>1 n2];

  forEach unvisited n from n1 with (3){

    print(1);

  }

}
```

7 Lessons Learned

7.1 Advice to future teams

7.1.1 Start early

As is the case on any design project, team GRAPL only gradually became aware of imperfections in both the design of the language and the implementation of the grapl-ast to java-ast compiler. (For example, the handling of variable declarations as in separate node in the ast, rather than as a type of statement, created complexities in the compiler.) In a real-world development environment, the current product would be only a prototype, and a final product would be built from the ground up to sidestep the issues that were discovered along the way. (Of course, on a large project, even more iterations would be performed.) Naturally, the pace of the course did not allow for this; however, had we gotten more of a jump on the project, we might have been able to revise the compiler incrementally.

The authors find that this particular lesson needs to be learned many, many times before it sticks.

7.1.2 Understand MicroC

A complete understanding of the microc compiler is recommended before writing your own compiler. It is explained in class, but takes time to digest. This will ultimately produce good component and interface design, wchich results in better division of labor between team members. We understood most, but not all, of what was going on in this compiler, and later found that we had reinvented the wheel in some places.

7.1.3 Synchronize your operating systems

Team members had varying degrees of ability to run executables natively or on Unix emulators as well as inside the Eclipse IDE. This made it difficult to share information quickly.

7.1.4 Use shell scripts

Though we always intended to automate the testing process by revising the existing Microc shell script, a combination of mishaps delayed delivery of this product until it was no longer practical to use it. We would also have benefited from simple scripts to run the compiler and produce a correctly named Java file.

7.1.5 Lock down file names and folder hierarchy

Though we did all of our file-sharing through SVN, we did not specify folder names and hierarchies strictly, and the changes to these along the way made committing and updating files—as well as simply locating the current test files and Java backend classes—needlessly difficult.

7.1.6 Synchronize data structures whenever possible

Inconsistencies in names and data structures between modules slowed development. In particular, declarations in the grapl-ast were stored as a (type, id) tuple; in the java-ast, for use with List lookup functions, they are converted to a tuple of (id, type). Because both the grapl-ast and the java-ast tuples appear in several places in the compiler, this simple flipping of the tuple created trivial but highly annoying headaches. We would have been much better served to revise the grapl-ast for consistency.

7.2 Team Member Statements

7.2.1 Lili Chen

This project makes me understand more about compiler. Though we learned a lot of knowledge in the class, when we are facing to design our own language, it is a challenging work. As far as I am concerned, the following three aspects are most important for me:

First, we should understand the principles of compiler deeply before implementing our project. In order to know much more about it, we should not only focus on the lectures related to it, but also analysis the micro C example provided by professor. It is a nice example for us to know the real world design and implement.

Secondly, in order to implement the project, OCaml is really important but new language for us. I feel difficult to learn it at first. However, I feel it is really useful afterwards.

Finally, effective communications among team members is necessary for the teamwork. We can always conclude the best ideas after our discussions and meetings.

7.2.2 Andres Uribe

I learned the basic process of how a decent compiler should work and the ins and outs of doing semantic checking. I learned a new programming paradigm with O'Caml. I also learned that for any problem when implementing the converter we could just defer the problem to the next component. Ultimately everything could be solved in Java, but we tried to avoid this and to do the right work in the right places. Finally, I'd recommend two things for future teams. First, understand the microc compiler before writing your own compiler. It is explained in class, but takes time to digest. This will ultimately produce good component and interface design, wchich results in better division of labor between team members. Second, write a substantial number of sample programs that your language is intended to work for before writing the LRM.

7.2.3 Ryan Turner

It goes without saying that I learned a heck of a lot about syntax and regular expressions and about issues of language design, especially scoping rules and semantic analysis. Probably more importantly for my future in programming, I came to really appreciate and enjoy the power of functional programming, What was a totally alien style of thought and syntax at the beginning of the semester now comes to me pretty easily. This being one of

the larger and longer programming projects I've worked on, I also came to see the value of an iterative development approach; because it simply isn't possible to anticipate all the consequences of a design decision, missteps are inevitable.

At the risk of kissing a little too much ass, I really had a blast on this project. O'Caml is certainly the most interesting language I've learned in a while, and the discussions and debates we had over issues of language and compiler design were the liveliest and most engaging that I've enjoyed to date in my career in computer science. Would that rest of my coursework were more like this.

7.2.4 Di Wen

1) I should have spent more time on practice on O'Caml besides this project.

Familiarity with O'Caml will greatly improve the understanding of the project; however, to become familiar with a new language do take a lot of time, especially for O'caml because this language's syntax and structure is quite different from some languages we're used to: C, JAVA, etc.

2) I should have paid more attention to what other group members are doing.

This is an integrated project. Having full knowledge on other members' parts will benefit working on one's own part because all parts will finally be connected.

7.2.5 Yi Yang

In this course I first learned about the process of building a compiler, which is obvious. We implemented a lot of stuff, scanner, parser, abstract syntax tree, pretty printer, etc. By doing the project, it makes me understand the importance of semantic checking, and how hard it is to check the correct semantic, which I took for granted during my past programming experience. I think actually the semantic checking stuff took us much more time than other part.

During the project, I also learned the power of another functional language, I also do LISP for artificial intelligence for this semester. Sometimes I just got Ocaml and Lisp mixed up since they have a lot of in common. But LISP is based on lambda calculus, while Ocaml not. I'd like to say, to do functional programming, recursion is necessary. But at the beginning, those recursion caused a lot of headaches, but when I get used to it in the process of doing the project. Recursion reveals its charming, it's really elegant and make codes simple.

8 Appendix A – Code Listing

```
(* scanner.mll *)
(* @authors: Steven Edwards, Lili Chen, Di Wen *)

{ open Parser }

(* letter, digit, and definition of number in GRAPL *)
let letter = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let num = digit+ | digit*'.'digit+

rule token = parse
| "/*"    { comment lexbuf }
| [' ' '\t' '\r' '\n'] { token lexbuf }
| '(' { LPAREN }
| ')' { RPAREN }
| '{' { LBRACE }
| '}' { RBRACE }
| '[' { LBRACK }
| ']' { RBRACK }
| "!=" { NEQ }
| '>'  { GT }
| '<'  { LT }
| "<=" { LEQ }
| ">=" { GEQ }
| "==" { EQ }
| '='  { ASSIGN }
| '!'  { NOT }
| "++" { INC }
| "--" { DEC }
| "<<"  { LEDGE }
| ">>"  { REDGE }
| "<>"  { BEDGE }
| "::"  { CONS }
| "while" { WHILE }
| "graph" { GRAPH }
| "node"          { NODE }
| "list"          { LIST }
| "number"          { NUMBER }
| "boolean"          { BOOLEAN }
| "bool"          { BOOLEAN }
| "void"          { VOID }
| "if"            { IF }
| "then"          { THEN }
| "else"          { ELSE }
| "forEach"       { FOREACH }
| "foreach"       { FOREACH }
| "from"          { FROM }
| "to"            { TO }
| "with"          { WITH }
| "return"        { RETURN }
| "visited"         { VISITED }
| "visit"           { VISIT }
| "unvisit"         { UNVISIT }
| "isVisited"       { ISVISITED }
| "unVisited"       { UNVISITED }
```

```
| "unvisited"      { UNVISITED }
| ';' { SEMICOLON }
| ',' { COMMA }
| '+' { PLUS }
| '-' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }
| "&&" { AND }
| "||" { OR }
| num as s { LITERAL(float_of_string(s)) }
| letter (letter | digit | '_')* as identi { ID(identi) }
| eof { EOF }
| "true" { BOOLEAN_LITERAL("true") }
| "false" { BOOLEAN_LITERAL("false") }

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

```
**************************************************************************************************************
*******
/* parser.mly */
/* @authors: Di Wen, Andres Uribe, Ryan Turner */

%{ open Ast  %}

%token LBRACK RBRACK LPAREN RPAREN LBRACE RBRACE
%token SEMICOLON COMMA
%token PLUS MINUS TIMES DIVIDE AND OR
%token EQ NEQ LT LEQ GT GEQ NOT ASSIGN CONS INC DEC
%token VOID GRAPH NODE LIST NUMBER BOOLEAN
%token REDGE LEDGE BEDGE HEAD TAIL
%token IF THEN ELSE FOREACH WITH RETURN WHILE
%token FROM TO
%token VISITED ISVISITED UNVISITED VISIT UNVISIT
%token <string> ID
%token EOF
%token <float> LITERAL
%token <string> BOOLEAN_LITERAL

%nonassoc ELSE
%right ASSIGN
%right CONS
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT
%left INC
%left DEC

%start program
%type < Ast.program> program

%%

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

```
fdecl:
   types ID LPAREN formals_opt RPAREN LBRACE vdecls stmts RBRACE
     { { rettype = $1;
       fname = $2;
          formals = $4; (* a list of tuples *)
          locals = List.rev $7; (* a list of tuples *)
          body =  List.rev $8 (* a list of statements *)} }

types: /* note that graph is not a type */
        | NUMBER { "number" }
        | NODE   { "node" }
        | BOOLEAN { "boolean" }
        | LIST { "list" }
        | VOID { "void" }

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

formal_list:
    types ID                   { [($1 , $2)] }
  | formal_list COMMA types ID { ($3, $4) :: $1 }

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

vdecl:
        | types ID SEMICOLON { ($1, $2) }

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

stmt:
        | LBRACE stmts RBRACE { Block(List.rev $2) }
        | IF LPAREN expr RPAREN THEN stmt ELSE stmt { IfThenElse($3, $6, $8) }
        /*| IF LPAREN expr RPAREN THEN stmts %prec NOELSE { IfThenElse($3, $6, Block([])) }*/
        | FOREACH qualifiers ID fromTo ID withStmt stmt { Foreach($2, $3, $4, $5, $6, $7)}
        | expr SEMICOLON { Expr($1) }
        | GRAPH LBRACK consdecls_opt RBRACK SEMICOLON { Graph(List.rev $3)}
        | RETURN expr SEMICOLON { Return($2) }
        | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

qualifiers:
        | /* nothing */ { "null" }
        | VISITED  { "visited" }
        | UNVISITED{ "unvisited" }

fromTo:
        | FROM { "from" }
        | TO { "to" }

withStmt:
        | /*nothing */ {(Noop, Literal(1.0)) }
        | WITH LPAREN pred RPAREN { $3 }

pred:
        | EQ     expr { (Equal, $2) }
        | NEQ    expr { (Neq, $2) }
        | LT     expr { (Less, $2) }
        | LEQ    expr { (Leq, $2) }
        | GT     expr { (Greater, $2) }
        | GEQ    expr { (Geq, $2) }

consdecls_opt: /* a list of triples  */
  | /*nothing*/ { [] }
  | consdecl_list { $1 }
```

```
/*| consdecls COMMA consdecl { $3 @ $1 }*/

consdecl_list:
  | consdecl{ List.rev $1 }
  | consdecl_list COMMA consdecl { List.rev $3 @  $1 }

consdecl: /* a list of triples */
        | ID edgeDefs
                { let rec help a b = match b with
                        | [] -> [(a,NoEdge(0.0),a)]
                        | head :: tail -> (a, fst head, snd head) :: help (snd head) (tail) in help $1
(List.rev $2)
                }

edgeDefs: /* a list of tuples*/
         /*nothing*/ { [] }
        | edgeDefs edgeDef { $2::$1 }

edgeDef: /* a tuple of the form ( <<3 , b) */
        | REDGE ID { (Redge(1.0), $2) }
        | LEDGE ID { (Ledge(1.0), $2) }
        | BEDGE ID { (Bedge(1.0), $2) }
        | REDGE LITERAL ID { (Redge($2), $3) }
        | LEDGE LITERAL ID { (Ledge($2), $3) }
        | BEDGE LITERAL ID { (Bedge($2), $3) }

expr:
        | ID { Id($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 EQ     expr { Binop($1, Equal, $3) }
        | expr NEQ    expr { Binop($1, Neq,   $3) }
        | expr LT     expr { Binop($1, Less,  $3) }
        | expr LEQ    expr { Binop($1, Leq,   $3) }
        | expr GT     expr { Binop($1, Greater,  $3) }
        | expr GEQ    expr { Binop($1, Geq,   $3) }
        | expr CONS  expr  { Binop($1, Cons, $3) }
        | LITERAL          { Literal($1) }
        | BOOLEAN_LITERAL  { BooleanLiteral($1) }
        | LPAREN expr RPAREN { $2 }
        | ID ASSIGN expr   { Assign($1, $3) }
        | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
        | VISIT ID { Visit($2) }
        | UNVISIT ID { Unvisit($2) }
        | LBRACK node_ids RBRACK { List($2) }
        | expr INC { Unop(Inc, $1) }
        | expr DEC { Unop(Dec, $1) }
        | NOT expr { Unop(Not, $2) }

/* list of node ids */
node_ids:
        | /* nothing */ { [] }
        | ID { [Id($1)] }
        | ID COMMA node_ids { Id($1)::$3 }

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

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



*************************************************************************************************************
*******
```

```
(* ast.ml *)
(* An abstract syntax tree representing the GRAPL language *)
(* @authors Team GRAPL *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | Noop | Cons | Inc | Dec | Not

type expr =
  | Literal of float
        | BooleanLiteral of string
  | Id of string
  | Binop of expr*op*expr
        | Unop of op*expr
  | Assign of string*expr
  | Call of string*expr list
  | Unvisit of string
  | Visit of string
  | Noexpr
  | List of expr list

type edge =
        | Redge of float
        | Ledge of float
        | Bedge of float
        | NoEdge of float

type stmt =
  | Block of stmt list   (* Statements *)
  | Expr of expr                  (* { ... } *)
  | Foreach of string*string*string*string*(op*expr)*stmt (* foreach Unvisited/visited A from with B <10
{...}*)
  | While of expr*stmt
        | IfThenElse of expr*stmt*stmt (* if (foo == 42) then {} else {} *)
  | Graph of (string*edge*string) list (* graph [a >>4 b <>3 c << d];*)
        | Return of expr

type func_decl = {
        rettype: string;
    fname : string;
    formals : (string*string) list;
    locals : (string*string) list;
    body : stmt list;
  }

type program = (string*string) list * func_decl list


*************************************************************************************************************
*******
(* jast.ml *)
(* An abstract syntax tree representing Java constructs. More or less generic. *)
(* @authors Team GRAPL *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | Not | Inc | Dec | Noop

type expr =
        | Literal of float
        | BooleanLiteral of string
  | Id of string
  | Binop of expr*op*expr
        | Unop of op*expr
  | Assign of string*expr
  | Call of expr*string*expr list (* implicit-param * function-name * arg-list *)
  | Noexpr
  | List of expr list
  | StringLit of string

type stmt =
  | New of string*string*expr list (* e.g. n = new Node(); *)
        | NewDecl of string*string*expr list (* e.g. Node n = new Node(); *)
```

```
        | Decl of string*string (* e.g. number x *)
        | Block of stmt list     (* Statements *)
  | Expr of expr                 (* { ... } *)
  | Foreach of string*string*expr*stmt (* e.g. for (Node n : graph.getNode("a", "<", 6, "visited")) { stmt } *)
        | While of expr*stmt (* predicate * body *)
  | IfThenElse of expr*stmt*stmt (* e.g. if (foo == 42) then {} else {} *)
        | Return of expr
        | Nostmt

type method_decl = {
        jrettype: string;
    jfname : string;
    jformals : (string*string) list;
    jlocals : (string*string) list;
    jbody : stmt list;
  }

(* class-name * globals-list * method-decls-list) *)
type java_class = string * string list * method_decl list

(* imports-list * class-list *)
type program = string list * java_class list


**************************************************************************************************************
*******
(* gpp.ml *)
(* GRAPL Pretty Printer. Generates GRAPL code from a grapl-ast. For debugging purposes. *)
(* @author: Lili Chen *)

open Ast

let string_of_operator = function
        | Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
  | Equal -> "==" | Neq -> "!=" | Less -> "<" | Leq -> "<="
        | Greater -> ">" | Geq -> ">=" | Noop -> "" | Cons -> "::"
        | Not -> "!" | Inc -> "++" | Dec -> "--"

let rec string_of_expr = function
  | Literal(l) -> string_of_float l
        | BooleanLiteral(s) -> s
  | Id(s) -> s
  | Binop(e1, o, e2) ->
     string_of_expr e1 ^ " " ^ string_of_operator(o) ^ " " ^ string_of_expr e2
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
        | Call(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | Unvisit(e) -> "unvisit " ^ e
  | Visit(e) ->  "visit " ^ e
        | List(l) -> "[" ^ String.concat "," (List.map string_of_expr l) ^ "]"
        | Unop(Not, e) -> "!" ^ string_of_expr e
        | Unop(op, e) -> string_of_expr e ^ string_of_operator op

let string_of_pred = function
        | (Noop, Literal(1.0)) -> ""
        | (p,e) -> "with (" ^ string_of_operator(p) ^ " " ^ string_of_expr e ^ ")"

let string_of_edge = function
        | Redge(w) -> ">>" ^ string_of_float(w)
        | Ledge(w) -> "<<" ^ string_of_float(w)
        | Bedge(w) -> "<>" ^ string_of_float(w)
        | NoEdge(w) -> ""

let rec string_of_graph = function
        | [(n1, NoEdge(0.0), n2)] -> n1 ^ " "
        | (n1, NoEdge(0.0), n2) :: tail -> n1 ^ ", " ^ string_of_graph(tail)
        | (n1, edge, n2) :: tail -> n1 ^ " " ^ string_of_edge(edge) ^ " " ^ string_of_graph(tail)
        | [] -> ""
```

```
let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | IfThenElse(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
     string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2 ^ "\n"
        | Foreach(q,n1,ft,n2,(s,e),st) -> "foreach " ^ q ^ " " ^ n1 ^ " " ^ ft ^ " " ^ n2 ^ " " ^
string_of_pred((s,e)) ^ string_of_stmt(st) ^ "\n"
        | Graph(ses_list) -> "graph [" ^ string_of_graph(ses_list) ^ "];\n"
        | Return(e) -> "return " ^ string_of_expr e ^ ";\n"
        | While(e,s) -> "while (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s

let string_of_vdecl var = fst(var) ^ " " ^ snd(var) ^ ";\n"

let string_of_fdecl f =
        f.rettype ^ " " ^ f.fname ^ "(" ^ String.concat "," (List.map (fun t  -> fst t ^ " " ^ snd t)
f.formals) ^ ")\n{\n" ^
                String.concat "\n" (List.map (fun t  -> fst t ^ " " ^ snd t ^ ";") f.locals) ^ "\n" ^
String.concat "" (List.map string_of_stmt f.body) ^ "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)

**************************************************************************************************************
***
(* jpp.ml *)
(* Java Pretty Printer. Generates java code from a Java ast *)
(* @author: Lili Chen *)

open Jast

let default_keyword = "public static"
let default_implicit_param = "null"

let string_of_operator = function
        | Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
  | Equal -> "==" | Neq -> "!=" | Less -> "<" | Leq -> "<="
        | Greater -> ">" | Geq -> ">=" | Noop -> ""
        | Not -> "!" | Inc -> "++" | Dec -> "--"

let rec string_of_expr = function
  | Literal(l) -> string_of_float l
        |       BooleanLiteral(s) -> s
  | Id(s) -> s
        | Unop(Not, e) -> "!" ^ "(" ^ string_of_expr e ^ ")"
        | Unop(op, e) -> "(" ^ string_of_expr e ^ ")" ^ string_of_operator(op)
  | Binop(e1, o, e2) ->
     string_of_expr e1 ^ " " ^ string_of_operator(o) ^ " " ^ string_of_expr e2
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Noexpr -> "fix_me"
  | Call(f,f_Name,f_List) -> (if ((string_of_expr f) <> default_implicit_param) then ((string_of_expr f) ^
".") else "" ) ^
                        f_Name ^ "(" ^ String.concat "," (List.map string_of_expr f_List) ^ ")"
  | List(l) -> "[" ^ String.concat "," (List.map string_of_expr l) ^ "]"
  | StringLit(l) -> "\"" ^ l ^ "\""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(e) -> "return " ^ string_of_expr e ^ ";\n";
  | New(n,m,e_list) -> m ^ " = new " ^ n ^ "(" ^ String.concat "," (List.map string_of_expr e_list) ^ ");\n"
        | While(e, s) -> "while (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
        | NewDecl(t,id,e_list) -> t ^ " " ^ id ^ " = new " ^ t ^ "(" ^ String.concat "," (List.map
string_of_expr e_list) ^ ");\n"
        | Decl(t, id) -> t ^ " " ^ id ^ ";\n"
 (*  | If(e, s) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s *)
```

```
  | IfThenElse(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | Foreach(t, id, e, stmt) ->
      "for (" ^ t  ^ " " ^ id ^ " : " ^ string_of_expr e ^ ")\n" ^ string_of_stmt stmt ^ "\n"
    (* e.g. Foreach n : graph.getNode("a", "< 6", "visited") *)
        | Nostmt -> ""

let string_of_formal (t, id) = t ^ " " ^ id

let string_of_global vdecl = snd(vdecl) ^ " " ^ fst(vdecl) ^ ";"

let string_of_method_decl method_decl =
        default_keyword ^ " " ^ method_decl.jrettype ^ " " ^ method_decl.jfname ^ "(" ^
        (String.concat "," (List.map string_of_formal method_decl.jformals))
        ^ ")\n{\n\n" ^
        (String.concat "" (List.map string_of_stmt method_decl.jbody))
        ^ "}\n"

(* class-name * globals-list * method-decls-list *)
(* type java_class = string * string list * method_decl list *)
let string_of_jclass extra_string java_class =
  "\npublic class " ^
  (fun(x, y, z) -> x ^ "\n{\n" ^ String.concat "\n" (List.map string_of_global y) ^ "\n\n" ^
        String.concat "\n\n" (List.map string_of_method_decl z) ^ extra_string) java_class ^ "}\n\n"

(* e.g., import lib.*; *)
let string_of_imports import_list =
  String.concat ";\n" import_list ^ ";\n"

let string_of_program (import_list, java_class_list, extra_string) =
  "\n" ^ string_of_imports import_list ^ "\n" ^
  String.concat "\n\n" (List.map (string_of_jclass extra_string) java_class_list) ^ "\n\n\n"



**********************************************************************************************************
*******

(* translate.ml *)
(* Translates a GRAPL AST to a Java AST and performs semantic checking.*)
(* @authors Ryan Turner, Andres Uribe *)

type mode = Quiet | Verbose (* compiler directives *)

open Ast
open Jast

(* Function symbol table *)
module FuncMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end)

(* Exception types *)
exception ReturnException of string
exception VariableNotFoundException of string
exception FunctionNotFoundException of string
exception BinopException of string
exception VisitUnvisitException of string
exception OtherException of string
exception AssignmentException of string
exception FunctionException of string
exception WhileExprException of string
exception ForeachNodeException of string
exception ForeachPredException of string
exception IfExprException of string
exception ReturnTypeException of string
exception MainNotFoundException of string
```

```
(* Default information for the backend java class that will run the GRAPL programs *)
(* let grapl_std_lib = GraplStdLib.string_of_library *)
let grapl_std_lib = "GraplStdLib"
let default_imports = ["lib.*"; grapl_std_lib] (* Java Backend classes *)
let default_lib = "library" (* Java Backend classes *)
let default_graph = "graph" (* Java Backend classes *)
let default_createGraph = "createEmptyGraph" (* Java Backend classes *)
let default_vars = [("GraplLib", default_lib); ("Graph", default_graph)] (* library must be created first *)
let default_keywords = "public static"
let null_implicit_param = "null"
let main_formals = [("String[]", "args")]
let main_rettype = "void"
let default_prefix = "gpl_" (* prefix ensures that backend Java identifiers cannot conflict with the user's
identifiers *)
let java_globals_prefix = ""

(* Built-in functions are implemented in Java *)
let built_in_functions = [
        { rettype = "number"; fname = "length"; formals = [("list", "l")]; locals = []; body = [] } ;
        { rettype = "node"; fname = "head"; formals = [("list", "l")]; locals = []; body = [] } ;
        { rettype = "list"; fname = "tail"; formals = [("list", "l")]; locals = []; body = [] } ;
        { rettype = "void"; fname = "unvisitAll"; formals = []; locals = []; body = [] } ;
        { rettype = "boolean"; fname = "isVisited"; formals = [("node", "n")]; locals = []; body = [] } ;
        ]

(* Library functions are written in GRAPL and defined in the GRAPL Standard Library. The user can extend *)
(* the standard library as needed and record the function definitions here. *)
let library_functions = [
        { rettype = "number"; fname = "numChildren"; formals = [("node", "n")]; locals = []; body = [] } ;
        { rettype = "number"; fname = "numAncestors"; formals = [("node", "n")]; locals = []; body = [] } ;
        { rettype = "void"; fname = "unvisitAllFrom"; formals = [("node", "n")]; locals = []; body = [] } ;
        { rettype = "void"; fname = "visitAllFrom"; formals = [("node", "n")]; locals = []; body = [] } ;
        { rettype = "list"; fname = "dfs"; formals = [("node", "start"); ("node", "finish")]; locals = [];
body = [] } ;
        { rettype = "list"; fname = "reverse"; formals = [("list", "old_lidt")]; locals = []; body = [] } ;

        ]

let libTable_data = (* Put standard library function declarations in a symbol table *)
                        List.fold_left (fun funcs fdecl -> FuncMap.add fdecl.fname fdecl funcs)
      FuncMap.empty (library_functions)

let initialize_standard_library = Jast.Expr(Jast.Call(Jast.Id(grapl_std_lib), "initialize",
[Jast.Id(default_graph); Jast.Id(default_lib)]))

(* Functions to extract the correct data structure from env tuple *)
(* env = (var_list, sp, funcTable, libTable, mode) *)
let var_list (v, s, f, l, m) = v
let sp (v, s, f, l, m) = s
let funcTable (v, s, f, l, m) = f
let libTable (v, s, f, l, m) = l
let mode (v, s, f, l, m) = m

(* Utility functions for pushing and popping local vars into symbol table var_list *)
(* var_list is the symbol table; sp keeps track of how many locals are currently in scope *)
let push_symbols env vars =
        let var_list = ((List.append vars (var_list env)))
        and sp = ((List.length vars) :: (sp env))
        in (var_list, sp, funcTable env, libTable env, mode env)

let pop sp = ((List.hd sp)-1)::(List.tl sp)
let rec pop_sym var_list sp = match var_list with
  | []-> [] (* failwith "empty" *)
  | hd::tl-> if(List.hd sp)>0 then pop_sym tl (pop sp) else hd::tl
let pop_symbols env = ((pop_sym (var_list env) (sp env)), (List.tl (sp env)), funcTable env, libTable env,
mode env)

(* Other utility functions *)
let flip_append_list string_tuple_list = List.map (fun (t, id) -> (default_prefix ^ id, t)) string_tuple_list
```

```
let flip (x, y) = (y, x)
let flip_list string_tuple_list = List.map flip string_tuple_list
let rec string_of_var_list var_list =
        let string_of_tuple (t, id) = "(" ^ t ^ "," ^ id ^ ")" in
        string_of_tuple (List.hd var_list) ^ (if (List.length var_list = 1) then "" else string_of_var_list
(List.tl var_list))
let append_keywords_list string_tuple_list = List.map (fun (id, t) -> (id, default_keywords ^ " " ^ t))
string_tuple_list

let java_type_for_type t = match t with
        | "node" -> "Node"
        | "graph" -> "Graph"
        | "number" -> "double"
        | "list" -> "List"
        | "void" -> "void"
        | "boolean" -> "boolean"
        | _  -> ""

let java_operator_for_operator o = match o with
        | Ast.Add -> Jast.Add
        | Ast.Sub -> Jast.Sub
        | Ast.Mult -> Jast.Mult
        | Ast.Div -> Jast.Div
        | Ast.Equal -> Jast.Equal
        | Ast.Neq -> Jast.Neq
        | Ast.Greater -> Jast.Greater
        | Ast.Geq -> Jast.Geq
        | Ast.Less -> Jast.Less
        | Ast.Leq -> Jast.Less
        | Ast.Noop -> Jast.Noop
        | Ast.Cons -> Jast.Noop
        | Ast.Not -> Jast.Not
        | Ast.Inc -> Jast.Inc
        | Ast.Dec -> Jast.Dec

let node_name_for_string s = default_prefix ^ s

(*********************************************************)
(* SEMANTIC CHECKING *)

let print_warning mode w = match mode with
        | Verbose -> print_string("/* WARNING: " ^ w ^ "*/\n")
        | Quiet -> print_string ""

(* Translating functions *)

(* Returns the type for a grapl expression, or raises exception for an invalid type. *)
let rec get_type_for env e = match e with
        | Ast.Literal(l) -> "number"
        | Ast.BooleanLiteral(s) -> "boolean"
        | Ast.Call(f, e_list) ->
         (let fdecl = try FuncMap.find f (funcTable env)
                with Not_found -> raise(FunctionNotFoundException(f ^" has not been declared"))
      in fdecl.rettype)
        | Ast.Noexpr -> "noexpr"
        | Ast.Unvisit(e) -> "node"
        | Ast.Visit(e) -> "node"
        | Ast.List(l) -> "list"
        | Ast.Unop(op, e) -> get_type_for env e
        | Ast.Binop(e1, op, e2) -> (match op with
                | Ast.Equal | Ast.Neq | Ast.Less | Ast.Leq | Ast.Greater | Ast.Geq -> "boolean"
                                                        | _ -> get_type_for env e2)
        | Ast.Assign(v, e) -> get_type_for env e

        (* Implicit global declaration of nodes inside the "graph" statement means that nodes are created
dynamically. The compiler *)
        (* cannot guarantee in all cases that a reference to an implicitly declared node will be valid; thus a
warning is generated. *)
        | Ast.Id(s) ->
```

```
          (try (List.assoc s (var_list env))
                        with Not_found -> print_warning (mode env) ("reference to implicitly declared node
\'" ^ s ^ "\' may not be valid at run-time."); "node" )

(* returns boolean after checking *)
let rec check_expr env e =
        match e with
    | Ast.Id(s) -> ((List.mem_assoc s (var_list env)) || ((get_type_for env (Ast.Id(s))) = "node"))
                | Ast.Unop(Ast.Not, e) -> get_type_for env e = "boolean"
                | Ast.Unop(op, e) -> get_type_for env e = "number"
                | Ast.Binop(e1,o,e2) ->
     if (check_expr env e1 && check_expr env e2)
                        then
                        (match o with
                                | Ast.Cons -> get_type_for env e1 = "node" && get_type_for env e2 = "list"

                                | _  -> ((get_type_for env e1) = (get_type_for env e2))
                                )
                        else
                                false
    | Ast.Assign(v,e) -> if (check_expr env e)
                                then
                                        (get_type_for env (Ast.Id(v))) = (get_type_for env e)
                                else
                                        false
    | Ast.Call("print", e_list) -> true

    | Ast.Call(f, e_list) -> (FuncMap.mem f (funcTable env)) || (FuncMap.mem f (libTable env)) (* TODO: check
argument list *)
    | Ast.List(e_list) -> List.fold_left (fun a e -> a && (check_expr env e)) true e_list
    | Ast.Literal(f) -> true
                | Ast.BooleanLiteral(s) -> true
    | Ast.Unvisit(n) -> true
    | Ast.Visit(n) -> true
    | Ast.Noexpr  -> true

(* Raises an exception for an invalid expression *)
let consistent_expr env e =
  let inner = check_expr env e
  in
  if (inner) then true
  else match e with
    | Ast.Id(s) -> raise(VariableNotFoundException("The variable \'" ^ s ^ "\' was not declared or is out of
scope."))
    | Ast.Binop(e1,op,e2) -> raise(BinopException("Problem with the binary expression: \'" ^
Gpp.string_of_expr(Ast.Binop(e1,op,e2)) ^ "\'."))
    | Ast.Unop(op,e2) -> raise(BinopException("Problem with the unary expression: \'" ^
Gpp.string_of_expr(Ast.Unop(op,e)) ^ "\'."))
                | Ast.Assign(v,e1) -> raise(AssignmentException("Type error: \'" ^ v ^ "\' cannot be assigned
to " ^ Gpp.string_of_expr(e1) ^ "\'."))
    | Ast.Call(f, e_list) -> raise(FunctionNotFoundException("The function \'"^ f ^"\' has not been
declared."))
    | Ast.List(e_list) -> raise(OtherException("Undefined exception in list."))
    | Ast.Literal(f) -> raise(OtherException("Undefined exception in literal \'" ^ string_of_float f ^ "\'."))
    | Ast.BooleanLiteral(s) -> raise(OtherException("Undefined exception in boolean \'" ^ s ^ "\'."))
                | Ast.Unvisit(n) -> raise(OtherException("Cannot call function 'unvisit' on non-node \'" ^ n ^
"\'."))
    | Ast.Visit(n) -> raise(OtherException("Cannot call function 'visit' on non-node \'" ^ n ^ "\'."))
    | Ast.Noexpr  -> raise(OtherException("Undefined exception in Noexpr."))


(*****************************************************************************)
(* CONVERTING AST --> JAST (Java-Ast) *)

(* Returns a Jast.expr for a given Ast.expr *)

let rec jast_of_expr env e =
        let jast_e expr = jast_of_expr env expr in
        if not(consistent_expr env e) then
```

```
                Jast.Noexpr (* invalid expressions should throw exceptions before reaching this point *)
        else
                match e with
                | Ast.Literal(l) -> Jast.Literal(l)
                | Ast.BooleanLiteral(s) -> Jast.BooleanLiteral(s)
          | Ast.Id(s) ->
                                if (not(List.mem_assoc s (var_list env)) && (get_type_for env (Ast.Id(s))) =
"node")
                                then Jast.Call(Jast.Id(default_graph), "getNode", [Jast.StringLit(s)])
                                else Jast.Id(default_prefix ^ s)
                | Ast.Unop(op, e) -> Jast.Unop(java_operator_for_operator op, jast_e e)
                | Ast.Binop(e1, Ast.Cons, e2) -> Jast.Call(jast_e e2, "addNew", [jast_e e1])
          | Ast.Binop(e1, op, e2) -> Jast.Binop(jast_e e1, (java_operator_for_operator op), jast_e e2)
          | Ast.Assign(v, e) -> Jast.Assign(default_prefix ^ v, jast_e e)
          | Ast.Noexpr ->  Jast.Noexpr
          | Ast.Unvisit(e) -> Jast.Call(jast_e (Ast.Id(e)), "unvisit", [])
          | Ast.Visit(e) ->  Jast.Call(jast_e (Ast.Id(e)), "visit", [])
          | Ast.List(l) -> Jast.Call(Jast.Id(default_lib), "buildList", List.map jast_e l)

                (* Built-in functions *)
          | Ast.Call("print", e_list) -> Jast.Call(Jast.Id(default_lib), "print", List.map jast_e e_list)
                | Ast.Call("unvisitAll", e_list) -> Jast.Call(Jast.Id(default_graph), "unvisitAll", [])
                | Ast.Call("length", e_list) -> let e = List.hd e_list in let x = get_type_for env e in
                                if (x = "list")
                                then Jast.Call(jast_e e, "size", [])
                                else raise (FunctionException("Error: Cannot call built-in 'length' on non-
list \'" ^ Gpp.string_of_expr(e) ^ "\'."))
                | Ast.Call("head", e_list) -> let e = List.hd e_list in let x = get_type_for env e in
                                if (x = "list")
                                then Jast.Call(Jast.Id(default_lib), "head", [jast_e e])
                                else raise (FunctionException("Error: Cannot call built-in 'head' on non-
list \'" ^ Gpp.string_of_expr(e) ^ "\'."))
                | Ast.Call("tail", e_list) -> let e = List.hd e_list in let x = get_type_for env e in
                                if (x = "list")
                                then Jast.Call(Jast.Id(default_lib), "tail", [jast_e e])
                                else raise (FunctionException("Error: Cannot call built-in 'tail' on non-
list \'" ^ Gpp.string_of_expr(e) ^ "\'."))
                | Ast.Call("isVisited", e_list) -> let e = List.hd e_list in let x = get_type_for env e in
                                if (x = "node")
                                then Jast.Call(Jast.Id(default_lib), "isVisited", [jast_e e])
                                else raise (FunctionException("Error: Cannot call built-in 'isVisited' on
non-node \'" ^ Gpp.string_of_expr(e) ^ "\'."))

                (* Other functions. Check the locally-declared functions first; this way, the user can
override library functions if desired.*)
                (* If the function is not found in the local function table, it must be a library function
(since it is a valid expression). *)
                | Ast.Call(f, e_list) ->
                                if (FuncMap.mem f (funcTable env))
                                then Jast.Call(Jast.Id(null_implicit_param), f, List.map jast_e e_list) (*
user-defined functions *)
                                else Jast.Call(Jast.Id(grapl_std_lib), f, List.map jast_e e_list) (*
standard library functions *)

(* Returns a function call to create a new edge in the graph *)
let jast_of_edge_def edge_def = match edge_def with
        | (s1, Ast.Redge(w), s2) ->
                        Jast.Expr(Jast.Call(Jast.Id("graph"), "addEdge", [Jast.StringLit(s1);
Jast.StringLit(s2); Jast.Literal(w)]))
        | (s1, Ast.Ledge(w), s2) ->
                        Jast.Expr(Jast.Call(Jast.Id("graph"), "addEdge", [Jast.StringLit(s2);
Jast.StringLit(s1); Jast.Literal(w)]))
        | (s1, Ast.Bedge(w), s2) ->
                        Jast.Expr(Jast.Call(Jast.Id("graph"),"addBEdge", [Jast.StringLit(s1);
Jast.StringLit(s2); Jast.Literal(w)]))
        | (s1, Ast.NoEdge(w), s2) -> Jast.Nostmt

(* Returns a Jast.stmt list ** NOT a Jast.stmt ** for each Ast.stmt *)
let rec jast_of_stmt rettype env stmt =
```

```
        let jast_s s = jast_of_stmt rettype env s in
        (*let jfold s_list = List.fold_left (jast_of_stmt env) s_list in*)
        let rec jast_of_foreach env (q, n1, ft, n2, (op,e), s) =
                let env = push_symbols env [(n1, "node")] in
                        let jast_e e = jast_of_expr env e in
                        match ft with
                                | "from" ->
                                                [Jast.Foreach("Node",

        default_prefix ^ n1,

        Jast.Call(Jast.Id("graph"),

                                        "getNodesFromWith",

                                        [ (jast_e (Ast.Id(n2)));

                                          Jast.Id("null");

                                                Jast.StringLit(q);

                                                Jast.StringLit(Gpp.string_of_operator(op));

                                                jast_of_expr env e

                                        ]),

        Jast.Block(jast_of_stmt rettype env s)

        )
                                                ]
                                | "to" ->
                                                [Jast.Foreach("Node",

        default_prefix ^ n1,

        Jast.Call(Jast.Id("graph"),

                                        "getNodesToWith",

                                        [ (jast_e (Ast.Id(n2)));

                                          Jast.Id("null");

                                                Jast.StringLit(q);

                                                Jast.StringLit(Gpp.string_of_operator(op));

                                                jast_of_expr env e

                                        ]),

        Jast.Block(jast_of_stmt rettype env s)

        )
                                                ]

                                | _ -> [Jast.Nostmt]
        in
                let jast_e e = jast_of_expr env e in
                        match stmt with
                | Ast.Expr(e) -> [Jast.Expr(jast_e e)]
                | Ast.Graph(edge_defs) -> List.map jast_of_edge_def edge_defs
                | Ast.Block(stmts) -> [Jast.Block(List.fold_left (fun a b -> a @ b) [] (List.map
(jast_of_stmt rettype env) stmts ))]
                | Ast.Return(e) -> if (not(rettype = get_type_for env e))
                        then raise(ReturnTypeException("Error: Return type of the function doesn't match with
the return type in \'"^Gpp.string_of_expr e^"\'."))
             else [Jast.Return(jast_e e)]
```

```
          | Ast.IfThenElse(e, s1, s2) ->
            if (not(  get_type_for env e = "boolean") )
            then raise(IfExprException("Error: Expresion inside the if clause must evaluate to a boolean in
\'"^ Gpp.string_of_expr e^"\'."))
            else [Jast.IfThenElse(jast_e e, Jast.Block(jast_s s1), Jast.Block(jast_s s2))]
          | Ast.Foreach(q,n1,ft,n2,(o,e),s) ->
      if(not( get_type_for env (Ast.Id(n2)) = "node"))
      then raise(ForeachNodeException("Error: Identifier after the from/to has to resolve to a node type." ))
      else if (not( get_type_for env e = "number" ))
        then raise(ForeachPredException ("Error: Predicate expression: \'"^Gpp.string_of_expr e^"\'. doesn't
evaluate to a number inside foreach loop"))
      else jast_of_foreach env (q, n1, ft, n2, (o,e), s)
        | Ast.While(e,s) ->
     (* type checking *)
     if (get_type_for env e = "boolean") then
       [Jast.While(jast_e e, Jast.Block(jast_s s))]
     else
       raise(WhileExprException("Error: Expresion inside the while loop has to evaluate to a boolean in
\'"^Gpp.string_of_expr e^"\'."))

(* Works on Java types. Java primitives do not need to be instantiated; objects do. *)
let instantiate prefix (id, t) = match t with
        | "Node" -> Jast.New(t, prefix ^ id, [Jast.StringLit(id)])
        | "List" -> Jast.New(t, prefix ^ id, [])
        | "GraplLib" -> Jast.New(t, prefix ^ id, [])
        | "Graph" -> Jast.Expr(Jast.Assign(default_graph, Jast.Call(Jast.Id(default_lib), default_createGraph,
[])))
        | _  -> Jast.Decl(java_type_for_type t, prefix ^ id)

(* Works on grapl types only *)
let declare_and_instantiate prefix (t, id) = match t with
        | "node" -> Jast.NewDecl(java_type_for_type t, prefix ^ id, [Jast.StringLit(id)])
        | "list" -> Jast.NewDecl(java_type_for_type t, prefix ^ id, [])
        | _  -> Jast.Decl(java_type_for_type t, prefix ^ id)

(* Returns a Jast.method_decl for each function in the grapl file *)
let jast_of_function globals_list env func =
                {
                        jrettype = if func.fname = "main" then main_rettype else java_type_for_type
func.rettype;
                        jfname = func.fname;
                        jformals =
                                if func.fname = "main"
                                then main_formals
                        else List.map (fun (t, id) -> (java_type_for_type t, default_prefix ^ id) )
func.formals;
                        jlocals = List.map (fun (t, id) -> (java_type_for_type t, default_prefix ^ id) )
func.locals;
                (* push extra initializations when doing the main *)
                        jbody =
                                let env = push_symbols env ((List.map flip func.locals) @ (List.map flip
func.formals))
                                in
                if (func.fname = "main")
                                then
                                        (List.map (instantiate java_globals_prefix) globals_list) @
                                        (List.map (declare_and_instantiate default_prefix) func.locals) @
                                        ([initialize_standard_library]) @
                                        (List.fold_left (fun a b -> a @ b) [] (List.map (jast_of_stmt
func.rettype env) func.body))
                                else
                                        (List.map (declare_and_instantiate default_prefix) func.locals) @
                                        (List.fold_left (fun a b -> a @ b) [] (List.map (jast_of_stmt
func.rettype env) func.body))
                        }

(* Where the magic happens *)
let translate vars funcs program_name mode_data =
        let ft = (* Put function declarations in a symbol table *)
```

```
                        (List.fold_left (fun funcs fdecl -> FuncMap.add fdecl.fname fdecl funcs)
      FuncMap.empty (funcs @ built_in_functions) )
         in let env = ([], [], ft, libTable_data, mode_data)
         in let env = push_symbols env (List.map flip vars) (* returns updated variable symbol table & stack-
pointer list *)
         in let jast_of_program env globals_list func_list =

                        (* Java program is a tuple (imports-list, class-list, default_lib) *)
                        let globals = (flip_append_list (List.map (fun (t,id) -> (java_type_for_type t, id))
globals_list)) @ (flip_list default_vars)
                        in
                        if not(FuncMap.mem "main" (funcTable env))
         then raise(MainNotFoundException("Error: main function was not found in the input file"))
                        else [(program_name, append_keywords_list globals, List.map (jast_of_function globals
env) func_list)]

 in jast_of_program env vars funcs


*************************************************************************************************************
*******

(* graplc.ml *)
(* @authors Steven Edwards, Ryan Turner, Di Wen *)
(* Runs the parser, scanner, and compiler on an input program.*)
(* Command line flags: *)
(* The following flags may be combined into a single argument in any order, e.g. -aq *)
(* -a : grapl pretty printer output (for debugging purposes)*)
(* -c : compile to class GraplProgram (default setting). User should redirect this output to GraplProgram.java
*)
(* -n myClass : compile to class MyClass. User should redirect this output to MyClass.java. *)
(* -l : compile the standard library, GraplStdLib. User should redirect this output to GraplStdLib.java. NOTE:
the library *)
(* will not function correctly unless compiled with this setting. *)
(* -q : quiet mode -- suppresses warnings *)
(* -v : verbose mode (default setting) *)

(* Note: some of us still have issues with compiling the Standard Library. This may not work correctly. *)

type action = Ast | Compile | NamedCompile | LibraryCompile

let default_program_name = "GraplProgram"
let standard_library_name = "GraplStdLib"
let default_imports = ["import lib.*"]
let std_lib_imports = ["package lib"]
let std_lib_init_string = "void initialize(Graph g, GraplLib l) \n{\n graph = g; \n library = l; \n}\n"

let _ =
  if Array.length Sys.argv > 1 then let flags = Sys.argv.(1) in
    let mode =
                        if String.contains flags 'q' then Translate.Quiet else
                        if String.contains flags 'v' then Translate.Verbose else Translate.Verbose

                and action =
                        if String.contains flags 'a' then Ast else
                        if String.contains flags 'c' then Compile else
                  if String.contains flags 'n' then NamedCompile else
                        if String.contains flags 'l' then LibraryCompile else Compile
                in
  let lexbuf = Lexing.from_channel stdin in
  let (vars, funcs)  = Parser.program Scanner.token lexbuf in
  match action with

        (* Grapl pretty printer -- for debugging purposes *)
  | Ast -> let listing = Gpp.string_of_program (vars, funcs)
           in print_string listing

        (* Normal compile with default output name. User must redirect output to GraplProgram.java. *)
  | Compile -> let listing = Jpp.string_of_program (default_imports, (Translate.translate vars funcs
```

```
default_program_name mode), "")
               in print_string listing

        (* Compile with named output. User must redirect output to a Java file with same name as the argument.
*)
  | NamedCompile -> let listing = Jpp.string_of_program (default_imports, (Translate.translate vars funcs
(Sys.argv.(2))) mode, "")
               in print_string listing

        (* Library compile. Incoporates a dirty little hack that allows global params to be passed to the
standard library class. *)
  | LibraryCompile ->
                let listing = Jpp.string_of_program (std_lib_imports, (Translate.translate vars funcs
standard_library_name mode), std_lib_init_string)
    in print_string listing
```

```
/** GRAPL STANDARD LIBRARY ***************************/
* NOTE: not all tested.

/** visitAllFrom(node n)
/*  Visits all ancestors of a node
/*    @args node n
/*  @return void
/**/
void visitAllFrom(node n)
{
  forEach c from n
  {
    visit c;
    visitAllFrom(c);
  }
}

/** numChildren(node n)
/*  Counts the immediate children of a node
/*    @args node n
/*  @return number
/**/
number numChildren(node n)
{
  number x;
  x = 0;
  forEach c from n
  {
    x++;
  }
```

```
  return x;
}

/** numAncestors(node n)
/*  Counts the immediate children of a node
/*    @args node n
/*  @return number count
/**/
number numAncestors(node n)
{
  number count;
  count = 0;
  unvisitAll();

  return numAncestorsHelper(n,count);
}

number numAncestorsHelper(node n, number x)
{
  visit n;
  forEach unvisited c from n
  {
    return x + numAncestorsHelper(c, x);
  }
  return 0;
}

/** reverse(list l)
/*  Returns a reversed list
/*    @args list old_list
/*  @return list new_list
/**/
list reverse(list old_list)
{
      list new_list;
      number length;
      new_list = [];
      length = length(old_list);

      while (length > 1)
      {
            head(old_list) :: new_list;
            old_list = tail(old_list);
            length = length - 1;
```

```
      }

      new_list = head(old_list) :: new_list;
      return new_list;

}

/** visitAllFrom(node start)
/*  visits all nodes reachable from n
/*    @args node start
/*  @return void
/**/
void visitAllFrom(node start)
{
      visit start;

      forEach unvisited x from start
      {
            visitAllFrom(x);
      }
}

/** unvisitAllFrom(node start)
/*  unvisits all nodes reachable from n
/*    @args node start
/*  @return void
/**/
void visitAllFrom(node start)
{
      unvisit start;

      forEach visited x from start
      {
            unvisitAllFrom(x);
      }
}

/** dfs(node n)
/*  Finds the shortest path from a given node to another
/*    @args node start, node end
/*  @return list path
/**/
list dfs (node start, node finish) {
      list l;
```

```
      unvisitAll();
      return dfs_helper(start, finish,l);
}

list dfs_helper(node start, node finish, list l)
{
      visit start;
      if (start != finish) then
      {
            forEach unvisited n from start
            {
            n::l;
            if(n!=finish)then{
                  dfs_helper(n, finish,l);
                  }
                  else
                  {
                  return l;
                  }

            }
      }
      else
      {
      }
      return l;
}

void main() {} /* dummy function to avoid compiler error */
```

9 Appendix B – Java Backend Classes

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lib;


/**
 *
 * @author _yy
```

```
 */
public interface GraplFacade {
    Graph createEmptyGraph();
    Node head(List l);
    List tail(List l);
    List buildList(Node... n);
    void print(List l);
    void print(Node e);
    void print(Graph p);
    void print(boolean b);
    void print(double i);
    void print(String s);
    int lengthList(List l);

}



/**************************************************************************************************
******
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lib;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;

/**
 *
 * @author _yy
 */
public class Graph {

    private ArrayList<Node> nodes = new ArrayList<Node>();
    private ArrayList<Edge> edges = new ArrayList<Edge>();

    public Graph(String name) {
        this.addNode(new Node(name));
    }

    public Graph() {
    }

    public Graph(String[] args) {
        System.out.println(args);
    }

    //add a single Node into the graph
    public void addNode(Node e) {
        boolean ne1 = false;
        Iterator it = nodes.iterator();
        while (it.hasNext()) {
            Node n = (Node) it.next();
            if (n.getName().equals(e.getName())) {
//                System.out.println("The Node: " + e.getName() + " is already existed in the
Graph.");
                ne1 = true;
                break;
```

```
        }
    }
    if (ne1 == false) {
        nodes.add(e);
    }
}

public void addNode(String s) {
    boolean ne1 = false;
    Iterator it = nodes.iterator();
    while (it.hasNext()) {
        Node n = (Node) it.next();
        if (n.getName().equals(s)) {
            ne1 = true;
            break;
        }
    }
    if (ne1 == false) {
        nodes.add(new Node(s));
    }
}

public void visitAll() {
    Iterator it = nodes.iterator();
    while (it.hasNext()) {
        Node n = (Node) it.next();
        n.visit();
    }
}

public void unvisitAll() {
    Iterator it = nodes.iterator();
    while (it.hasNext()) {
        Node n = (Node) it.next();
        n.unvisit();
    }
}

public void addEdge(String n1, String n2, double weight) {
    addNode(n1);
    addNode(n2);
    //add new edge
    Edge e = new Edge(n1, n2, weight);
    edges.add(e);
}

public void addBEdge(String n1, String n2, double weight) {
    //add new edge
    addNode(n1);
    addNode(n2);
    Edge e = new Edge(n1, n2, weight);
    Edge e1 = new Edge(n2, n1, weight);
    edges.add(e);
    edges.add(e1);
}

public Node getNode(String e) {
    //check if the Node is in the graph
    Iterator it = nodes.iterator();
    while (it.hasNext()) {
```

```
            Node n = (Node) it.next();
            if (e.equals(n.getName())) {
                return n;
            }
        }
        return null;
    }

    public ArrayList<Node> removeDups(ArrayList<Node> l)
    {
        HashSet<Node> temp = new HashSet<Node>();
        temp.addAll(l);
        l.clear();
        l.addAll(temp);
        return l;
    }

    public ArrayList<Node> getNodesFromWith(String name, ArrayList<Node> path, String qualifier,
String predicate) {
        return removeDups(getNodesFromWith(getNode(name), path, qualifier, predicate));
    }

    public ArrayList<Node> getNodesFromWith(Node e, ArrayList<Node> path, String qualifier,
String predicate) {

        boolean visited = false;
        if (path == null) {
            path = new ArrayList<Node>();
        }
        if (qualifier.toLowerCase().equals("unvisited")) {
            visited = false;
        } else if (qualifier.toLowerCase().equals("visited")) {
            visited = true;
        } else {
            System.err.println("qualifier error.");
        }
        Iterator it = edges.iterator();
        if (predicate == null) {
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName())
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
            return path;
        } else if (predicate.contains("==")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName()) && ed.getEdgeweight() == inum
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
        } else if (predicate.contains("!=")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
```

```
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName()) && ed.getEdgeweight() != inum
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
        } else if (predicate.contains("<=")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName()) && ed.getEdgeweight() <= inum
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
        } else if (predicate.contains(">=")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName()) && ed.getEdgeweight() >= inum
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
        } else if (predicate.contains("<")) {
            String num = predicate.substring(1);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName()) && ed.getEdgeweight() < inum
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
        } else if (predicate.contains(">")) {
            String num = predicate.substring(1);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getStartNode().equals(e.getName()) && ed.getEdgeweight() > inum
                        && getNode(ed.getEndNode()).isVisit() == visited) {
                    path.add(getNode(ed.getEndNode()));
                }
            }
        }
        return path;
    }

    public ArrayList<Node> getNodesToWith(String name, ArrayList<Node> path, String qualifier,
String predicate) {
        return removeDups(getNodesToWith(getNode(name), path, qualifier, predicate));
    }

    public ArrayList<Node> getNodesToWith(Node e, ArrayList<Node> path, String qualifier, String
predicate, double weight) {
        if (predicate.equals("")) {
            return getNodesToWith(e, path, qualifier, null);
        } else {
```

```
            return getNodesToWith(e, path, qualifier, predicate + weight);
        }
    }

    public ArrayList<Node> getNodesFromWith(Node e, ArrayList<Node> path, String qualifier,
String predicate, double weight) {
        if (predicate.equals("")) {
            return getNodesFromWith(e, path, qualifier, null);
        } else {
            return getNodesFromWith(e, path, qualifier, predicate + weight);
        }
    }

    public ArrayList<Node> getNodesToWith(Node e, ArrayList<Node> path, String qualifier, String
predicate) {
        boolean visited = false;
        Iterator it = edges.iterator();

        if (path == null) {
            path = new ArrayList<Node>();
        }
        if (qualifier.toLowerCase().equals("unvisited")) {
            visited = false;
        } else if (qualifier.toLowerCase().equals("visited")) {
            visited = true;
        } else {
            System.err.println("qualifier error.");
        }

        if (predicate == null) {
            while (it.hasNext()) {

                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName())
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
            return path;
        } else if (predicate.contains("==")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName()) && ed.getEdgeweight() == inum
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
        } else if (predicate.contains("<=")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName()) && ed.getEdgeweight() <= inum
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
        } else if (predicate.contains(">=")) {
```

```
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName()) && ed.getEdgeweight() >= inum
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
        } else if (predicate.contains("!=")) {
            String num = predicate.substring(2);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName()) && ed.getEdgeweight() != inum
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
        } else if (predicate.contains("<")) {
            String num = predicate.substring(1);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName()) && ed.getEdgeweight() < inum
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
        } else if (predicate.contains(">")) {
            String num = predicate.substring(1);
            double inum = Double.parseDouble(num);
            while (it.hasNext()) {
                Edge ed = (Edge) it.next();
                if (ed.getEndNode().equals(e.getName()) && ed.getEdgeweight() > inum
                        && getNode(ed.getStartNode()).isVisit() == visited) {
                    path.add(getNode(ed.getStartNode()));
                }
            }
        }
        return path;
    }

    public int numNodes() {
        return nodes.size();
    }

    public int numEdges() {
        return edges.size();
    }

    public void infoGraph() {
        System.out.print("Graph: ");
        System.out.println("node: " + numNodes() + ", edge " + numEdges() + ".");
    }

    public void nameNodes() {
        Iterator it = nodes.iterator();
        System.out.println("This graph contains following Nodes:");
        while (it.hasNext()) {
```

```
            Node n = (Node) it.next();
            System.out.println(n.getName() + " ");
        }
    }

    public boolean Siblings(Node e) {
        boolean e1 = false;
        boolean e2 = false;
        Iterator it = nodes.iterator();
        while (it.hasNext()) {
            Node n = (Node) it.next();
            if (e.getName().equals(n.getName())) {
                e1 = true;
            }
        }
        if (e1 == false) {
            System.out.println("Node " + e.getName() + " is not found in the graph.");
        } else {
            Iterator it2 = edges.iterator();
            while (it2.hasNext()) {
                Edge ed = (Edge) it2.next();
                if (ed.getStartNode().equals(e.getName())) {
                    e2 = true;
                    System.out.println("Node: " + e.getName() + " connected to Node " +
ed.getEndNode() + " with weight " + ed.getEdgeweight());
                }
            }
        }
        if (e2 == false) {
            System.out.println("Node: " + e.getName() + " does not connect to any other node.");
            return false;
        } else if (e2 == true) {
            return true;
        }
        return false;
    }
//    public void add(Node n1, Node n2, String ed) {
//        String sub = ed.substring(0, 2);
//        double weight = Double.parseDouble(ed.substring(2));
////        int weight = Integer.parseInt(ed.substring(2));
//        if (sub.equals("<>")) {
//            addBEdge(n1, n2, weight);
//        } else if (sub.equals("<<")) {
//            addEdge(n1, n2, weight);
//        } else if (sub.equals(">>")) {
//            addEdge(n2, n1, weight);
//        } else {
//            System.out.println("Error initialize the graph");
//        }
//    }
}


/**********************************************************************************************
******
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package lib;
```

```java
import java.util.ArrayList;

/**
 *
 * @author _yy
 */
public class List extends ArrayList<Node>{

}
```

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

```java
package lib;

/**
 *
 * @author _yy
 */
public class Edge {

    private double edgeweight = 0;
    private String startNode = null;
    private String endNode = null;

    public Edge(String n1, String n2, double i) {
        this.startNode = n1;
        this.endNode = n2;
        this.edgeweight = i;
    }


    public double getEdgeweight() {
        return edgeweight;
    }

    public void setEdgeweight(int edgeweight) {
        this.edgeweight = edgeweight;
    }

    public String getEndNode() {
        return endNode;
    }

    public void setEndNode(String endNode) {
        this.endNode = endNode;
    }

    public String getStartNode() {
        return startNode;
    }

    public void setStartNode(String startNode) {
        this.startNode = startNode;
    }
}
```

```java
/**********************************************************************************************
******
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lib;

/**
 *
 * @author _yy
 */
public class Node {

    private String name;
    private boolean visit = false;

    public Node() {
    }

    public Node(String s) {
        this.name = s;
    }

    public String getName() {
        return name;
    }

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

    public boolean isVisit() {
        return visit;
    }

    public void visit() {
        this.visit = true;
    }

    public void unvisit() {
        this.visit = false;
    }
}
```

```java
/**********************************************************************************************
******
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lib;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;

/**
```

```
 *
 * @author _yy
 */
public class GraplLib implements GraplFacade {

    public Graph createEmptyGraph() {
        return (new Graph());
    }



    public void print(List l) {
        Iterator it = l.iterator();
        System.out.print("list: ");
        while (it.hasNext()) {
            Node n = (Node) it.next();
            System.out.print(n.getName() + " ");
        }
        System.out.println();
    }

    public void print(Node e) {
        if (e.isVisit()) {
            System.out.println("node " + e.getName() + ": visited.");
        }
        else
        {
            System.out.println("node " + e.getName() + ": not visited.");
        }
    }

    public List buildList(Node... n) {
        List l = new List();
        l.addAll(Arrays.asList(n));
        return l;
    }

    public void print(Graph p) {
        p.infoGraph();
    }

    public int lengthList(List l) {
        return l.size();
    }

    public void print(boolean b) {
        System.out.println(b);
    }

    public void print(double i) {
        System.out.println(i);
    }

    public void print(String s) {
        System.out.println(s);
    }

    public Node head(List l) {
        return (Node) l.get(0);
    }
```

```
    public List tail(List l) {
        Object temp = l.get(0);
        l.remove(temp);
        return l;
    }

}
```

10 Appendix C – Project Log

10.1.1.1 Today

- 3 hours ago

r263 (a little shell script to make and compile a file) committed by rct189 - a little shell script to make and compile a file

- 3 hours ago

r262 ([No log message]) committed by rct189 - [No log message]

- 3 hours ago

r261 ([No log message]) committed by rct189 - [No log message]

- 3 hours ago

r260 (upgrades to graplc) committed by rct189 - upgrades to graplc

- 21 hours ago

r259 (fixed pretty printer foreach case.) committed by andresuribe87 - fixed pretty printer foreach case.

- 21 hours ago

r258 ([No log message]) committed by rct189 - [No log message]

- 22 hours ago

r257 (renaming of files) committed by andresuribe87 - renaming of files

- 22 hours ago

r256 ([No log message]) committed by srct114 - [No log message]

- 22 hours ago

r255 ([No log message]) committed by srct114 - [No log message]

- 22 hours ago

r254 (Compiler with semantic checking added.) committed by andresuribe87 - Compiler with semantic checking added.

- 22 hours ago

r253 ([No log message]) committed by srct114 - [No log message]

- 22 hours ago

r252 (nomain submitted for testing.) committed by andresuribe87 - nomain submitted for testing.

- 22 hours ago

r251 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- 22 hours ago

r250 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- 22 hours ago

  r249 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 22 hours ago

  r248 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 22 hours ago

  r247 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 22 hours ago

  r246 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 22 hours ago

  r245 ([No log message]) committed by rct189 - [No log message]
- 22 hours ago

  r244 ([No log message]) committed by srct114 - [No log message]
- 22 hours ago

  r243 ([No log message]) committed by rct189 - [No log message]
- 23 hours ago

  r242 ([No log message]) committed by rct189 - [No log message]
- 23 hours ago

  r241 ([No log message]) committed by rct189 - [No log message]

10.1.1.2 Yesterday

- 32 hours ago

  r240 (std lib now pasted in as string in all grapl programs) committed by rct189 - std lib now pasted in as string in all grapl programs
- 33 hours ago

  r239 (some crazy std lib stuff) committed by rct189 - some crazy std lib stuff
- 33 hours ago

  r238 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 33 hours ago

  r237 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 34 hours ago

  r236 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 34 hours ago

  r235 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 34 hours ago

  r234 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 34 hours ago

  r233 (new std lib functions) committed by rct189 - new std lib functions
- 34 hours ago

  r232 ([No log message]) committed by qingfeng...@gmail.com - [No log message]
- 34 hours ago

  r231 ([No log message]) committed by rct189 - [No log message]

- 34 hours ago

r230 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- 35 hours ago

r229 ([No log message]) committed by rct189 - [No log message]

- 35 hours ago

r228 ([No log message]) committed by rct189 - [No log message]

- 35 hours ago

r227 ([No log message]) committed by rct189 - [No log message]

- 35 hours ago

r226 ([No log message]) committed by rct189 - [No log message]

- 36 hours ago

r225 (fixed variable declaration) committed by rct189 - fixed variable declaration

- 36 hours ago

r224 ([No log message]) committed by rct189 - [No log message]

- 36 hours ago

r223 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- 37 hours ago

r222 ([No log message]) committed by rct189 - [No log message]

- 37 hours ago

r221 ([No log message]) committed by rct189 - [No log message]

- 37 hours ago

r220 ([No log message]) committed by rct189 - [No log message]

- 37 hours ago

r219 ([No log message]) committed by rct189 - [No log message]

- 39 hours ago

r218 ([No log message]) committed by rct189 - [No log message]

- 39 hours ago

r217 ([No log message]) committed by rct189 - [No log message]

- 40 hours ago

r216 ([No log message]) committed by rct189 - [No log message]

- 40 hours ago

r215 ([No log message]) committed by rct189 - [No log message]

- 40 hours ago

r214 ([No log message]) committed by rct189 - [No log message]

10.1.1.3 Yesterday

- 42 hours ago

r213 ([No log message]) committed by rct189 - [No log message]

- 42 hours ago

r212 ([No log message]) committed by rct189 - [No log message]

- 42 hours ago

r211 ([No log message]) committed by rct189 - [No log message]

- 42 hours ago

r210 (string args for getfromwith) committed by rct189 - string args for getfromwith

- 42 hours ago

r209 (built in functions) committed by rct189 - built in functions

- 43 hours ago

r208 (dfs fixed) committed by andresuribe87 - dfs fixed

- 43 hours ago

r207 ([No log message]) committed by rct189 - [No log message]

- 43 hours ago

r206 (dfs test added) committed by andresuribe87 - dfs test added

- 43 hours ago

r205 ([No log message]) committed by rct189 - [No log message]

- 43 hours ago

r204 (Added two std lib functions.) committed by andresuribe87 - Added two std lib functions.

- 44 hours ago

r203 ([No log message]) committed by chenlili0603 - [No log message]

- 44 hours ago

r202 ([No log message]) committed by chenlili0603 - [No log message]

- 45 hours ago

r201 (added booleans and while loop) committed by rct189 - added booleans and while loop

- 45 hours ago

r200 ([No log message]) committed by srct114 - [No log message]

- 45 hours ago

r199 ([No log message]) committed by srct114 - [No log message]

- 45 hours ago

r198 ([No log message]) committed by srct114 - [No log message]

- 46 hours ago

r197 (final lib 16-04) committed by qingfeng...@gmail.com - final lib 16-04

- 46 hours ago

r196 ([No log message]) committed by srct114 - [No log message]

- 46 hours ago

r195 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- 46 hours ago

r194 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

10.1.1.4 Last 7 days

- Dec 20, 2010

r193 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 20, 2010

  r192 ([No log message]) committed by rct189 - [No log message]
- Dec 20, 2010

  r191 ([No log message]) committed by rct189 - [No log message]
- Dec 20, 2010

  r190 ([No log message]) committed by rct189 - [No log message]
- Dec 20, 2010

  r189 ([No log message]) committed by rct189 - [No log message]
- Dec 20, 2010

  r188 (addEdge now takes node as param instead of string) committed by rct189 - addEdge now takes node as param instead of string
- Dec 20, 2010

  r187 ([No log message]) committed by chenlili0603 - [No log message]
- Dec 20, 2010

  r186 ([No log message]) committed by chenlili0603 - [No log message]
- Dec 20, 2010

  r185 ([No log message]) committed by chenlili0603 - [No log message]
- Dec 20, 2010

  r184 ([No log message]) committed by chenlili0603 - [No log message]
- Dec 20, 2010

  r183 ([No log message]) committed by chenlili0603 - [No log message]
- Dec 19, 2010

  r182 (continued work on look_ahead) committed by rct189 - continued work on look_ahead
- Dec 19, 2010

  r181 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r180 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r179 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r178 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r177 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r176 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r175 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

  r174 ([No log message]) committed by srct114 - [No log message]
- Dec 19, 2010

r173 ([No log message]) committed by srct114 - [No log message]

- Dec 19, 2010

r172 ([No log message]) committed by srct114 - [No log message]

- Dec 19, 2010

r171 ([No log message]) committed by srct114 - [No log message]

- Dec 19, 2010

r170 ([No log message]) committed by srct114 - [No log message]

- Dec 19, 2010

r169 (latest) committed by qingfeng...@gmail.com - latest

- Dec 19, 2010

r168 (latest) committed by qingfeng...@gmail.com - latest

- Dec 19, 2010

r167 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r166 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r165 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r164 (latest java files) committed by qingfeng...@gmail.com - latest java files

10.1.1.5 Last 7 days

- Dec 19, 2010

r163 (Have updated fiendishly complex look_ahead_kludge. Truly an ...) committed by rct189 - Have updated fiendishly complex look_ahead_kludge. Truly an appalling design.

- Dec 19, 2010

r162 (added mind-blowing but extremely ugly look-ahead function to...) committed by rct189 - added mind-blowing but extremely ugly look-ahead function to extract all edgedefs from Graph statements ahead of time. Still doesn't fix the problem.

- Dec 19, 2010

r161 ([No log message]) committed by rct189 - [No log message]

- Dec 19, 2010

r160 (fixed global declaration) committed by rct189 - fixed global declaration

- Dec 19, 2010

r159 ([No log message]) committed by rct189 - [No log message]

- Dec 19, 2010

r158 (testShell) committed by srct114 - testShell

- Dec 19, 2010

r157 (Fixed various errors.) committed by andresuribe87 - Fixed various errors.

- Dec 19, 2010

r156 (mucking about with java) committed by rct189 - mucking about with java

- Dec 19, 2010

r155 (Added java foreach wrapper to accept separate operator and p...) committed by rct189 - Added java foreach wrapper to accept separate operator and predicate (expression) args

- Dec 19, 2010

r154 ([No log message]) committed by rct189 - [No log message]

- Dec 19, 2010

r153 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r152 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r151 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r150 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r149 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r148 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r147 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r146 (test) committed by qingfeng...@gmail.com - test

- Dec 19, 2010

r145 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r144 (Share project "ocamlBackup" into "https://grapl2010.googleco...) committed by qingfeng...@gmail.com - Share project "ocamlBackup" into "https://grapl2010.googlecode.com/svn"

- Dec 19, 2010

r143 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r142 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r141 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r140 (Share project "ocamlBackup" into "https://grapl2010.googleco...) committed by qingfeng...@gmail.com - Share project "ocamlBackup" into "https://grapl2010.googlecode.com/svn"

- Dec 19, 2010

r139 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r138 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r137 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r136 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r135 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r134 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r133 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r132 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r131 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r130 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r129 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r128 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r127 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r126 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r125 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r124 ([No log message]) committed by qingfeng...@gmail.com - [No log message]

- Dec 19, 2010

r123 (Added annotations) committed by andresuribe87 - Added annotations

- Dec 19, 2010

r122 (Added global node.) committed by andresuribe87 - Added global node.

- Dec 19, 2010

r121 (Added node declaration.) committed by andresuribe87 - Added node declaration.

- Dec 19, 2010

r120 (Exceptions may need some work.) committed by andresuribe87 - Exceptions may need some work.

- Dec 19, 2010

r119 (Added a few exceptions.) committed by andresuribe87 - Added a few exceptions.

- Dec 19, 2010

r118 (working with exceptions) committed by andresuribe87 - working with exceptions

- Dec 19, 2010

r117 (Added main cases.) committed by andresuribe87 - Added main cases.

- Dec 19, 2010

r116 (Fixed prefixes. Added annotations.) committed by andresuribe87 - Fixed prefixes. Added annotations.

- Dec 19, 2010

r115 (Added string literal to jast) committed by andresuribe87 - Added string literal to jast

- Dec 19, 2010

r114 (Fixed edgedefs) committed by andresuribe87 - Fixed edgedefs

10.1.1.6 Last 7 days

- Dec 19, 2010

r113 (Added printer for global variables.) committed by andresuribe87 - Added printer for global variables.

- Dec 19, 2010

r112 (backup) committed by qingfeng...@gmail.com - backup

- Dec 18, 2010

r111 (up-to-date. Refixed a bunch of stuff we already fixed. every...) committed by rct189 - up-to-date. Refixed a bunch of stuff we already fixed. everything working, but compile has some issues.

- Dec 18, 2010

r110 ([No log message]) committed by srct114 - [No log message]

- Dec 18, 2010

r109 (Cleaning files) committed by rct189 - Cleaning files

- Dec 18, 2010

r108 (A fixed jast) committed by rct189 - A fixed jast

- Dec 18, 2010

r107 (A compilable compiler.) committed by rct189 - A compilable compiler.

- Dec 18, 2010

r106 (Makefile linking fixed) committed by rct189 - Makefile linking fixed

- Dec 18, 2010

r105 (Trying to figure out) committed by rct189 - Trying to figure out

- Dec 18, 2010

r104 ([No log message]) committed by srct114 - [No log message]

- Dec 18, 2010

r103 ([No log message]) committed by srct114 - [No log message]

- Dec 18, 2010

r102 ([No log message]) committed by srct114 - [No log message]

- Dec 18, 2010

r101 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r100 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r99 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r98 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r97 ( ) committed by srct114 -

- Dec 18, 2010

r96 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r95 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r94 (check_expr added) committed by andresuribe87 - check_expr added

- Dec 18, 2010

r93 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r92 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r91 (latest) committed by qingfeng...@gmail.com - latest

- Dec 18, 2010

r90 (latest) committed by qingfeng...@gmail.com - latest

- Dec 18, 2010

r89 ([No log message]) committed by chenlili0603 - [No log message]

- Dec 18, 2010

r88 (dammit dammit dammit) committed by rct189 - dammit dammit dammit

- Dec 18, 2010

r87 ([No log message]) committed by srct114 - [No log message]

- Dec 18, 2010

r86 (latest) committed by qingfeng...@gmail.com - latest

- Dec 18, 2010

r85 (latest) committed by qingfeng...@gmail.com - latest

- Dec 18, 2010

r84 (perfect. everything working. go home.) committed by rct189 - perfect. everything working. go home.

- Dec 18, 2010

r83 ([No log message]) committed by rct189 - [No log message]

- Dec 18, 2010

r82 (working out the kinks in compile.ml. Still needs a lot of wo...) committed by rct189 - working out the kinks in compile.ml. Still needs a lot of work.

- Dec 17, 2010

r81 ([No log message]) committed by srct114 - [No log message]

- Dec 17, 2010

r80 ([No log message]) committed by srct114 - [No log message]

- Dec 17, 2010

r79 (First version of compile. Set return as a stmt. Fixed java e...) committed by rct189 - First version of compile. Set return as a stmt. Fixed java example.

- Dec 17, 2010

r78 (Return added.) committed by andresuribe87 - Return added.

- Dec 17, 2010

r77 (various stuffs) committed by rct189 - various stuffs

- Dec 17, 2010

r76 (Added return.) committed by andresuribe87 - Added return.

- Dec 17, 2010

r75 (Return added.) committed by andresuribe87 - Return added.

- Dec 17, 2010

r74 (syntax fixed) committed by andresuribe87 - syntax fixed

- Dec 17, 2010

r73 (Graph order fixed.) committed by andresuribe87 - Graph order fixed.

- Dec 17, 2010

r72 (parser shenanigans) committed by rct189 - parser shenanigans

- Dec 17, 2010

r71 (Graph fixed.) committed by andresuribe87 - Graph fixed.

- Dec 17, 2010

r70 (tests only) committed by rct189 - tests only

- Dec 17, 2010

r69 (edits to convert.ml) committed by rct189 - edits to convert.ml

- Dec 16, 2010

r68 (have a little problem implementing it, working on it.) committed by qingfeng...@gmail.com - have a little problem implementing it, working on it.

- Dec 16, 2010

r67 (started work on jast.ml (java-ast), convert.ml) committed by rct189 - started work on jast.ml (java-ast), convert.ml

10.1.1.7 Last 30 days

- Dec 11, 2010

r66 (an extremely preliminary stab at the java-ast builder. Locat...) committed by rct189 - an extremely preliminary stab at the java-ast builder. Located at bottom of ast file.

- Dec 11, 2010

r65 (Initial commit) committed by andresuribe87 - Initial commit

- Dec 11, 2010

r64 (Initial commit) committed by andresuribe87 - Initial commit

10.1.1.8 Last 30 days

- Dec 10, 2010

r63 (latest) committed by qingfeng...@gmail.com
latest

- Dec 10, 2010

r62 (latest) committed by qingfeng...@gmail.com
latest

- Dec 10, 2010

  r61 (latest) committed by qingfeng...@gmail.com
  latest

- Dec 10, 2010

  r60 (latest) committed by qingfeng...@gmail.com
  latest

- Dec 10, 2010

  r59 (test getNodeFrom function) committed by qingfeng...@gmail.com
  test getNodeFrom function

- Dec 10, 2010

  r58 (test getNodeFrom function) committed by qingfeng...@gmail.com
  test getNodeFrom function

- Dec 10, 2010

  r57 (add the way to create bi-direct edges) committed by qingfeng...@gmail.com
  add the way to create bi-direct edges

- Dec 10, 2010

  r56 (test it) committed by qingfeng...@gmail.com
  test it

- Dec 10, 2010

  r55 (test it) committed by qingfeng...@gmail.com
  test it

- Dec 10, 2010

  r54 (tested new java files and libs.) committed by qingfeng...@gmail.com
  tested new java files and libs.

- Dec 10, 2010

  r53 (tested new java files and libs.) committed by qingfeng...@gmail.com
  tested new java files and libs.

- Dec 10, 2010

  r52 (make clean) committed by rct189
  make clean

- Dec 10, 2010

  r51 (Misc updates. Renamed test extensions to .gpl and edited tes...) committed by rct189
  Misc updates. Renamed test extensions to .gpl and edited test files.

- Dec 09, 2010

  r50 (almost Hello World!) committed by rct189
  almost Hello World!

- Dec 09, 2010

  r49 ([No log message]) committed by srct114
  [No log message]

- Dec 08, 2010

  r48 ([No log message]) committed by srct114
  [No log message]

- Dec 08, 2010

r47 (Structured project with folders. Added sample java output. A...) committed by andresuribe87
Structured project with folders. Added sample java output. Added some GRAPL simple test cases.

- Dec 07, 2010

r46 (stmt/stmts bug fixed (per Prof. Edwards)) committed by rct189
stmt/stmts bug fixed (per Prof. Edwards)

- Dec 06, 2010

r45 ([No log message]) committed by rct189
[No log message]

- Dec 06, 2010

Test Cases.rtf (Test Cases) file uploaded by srct114
Labels: Test Cases

- Dec 06, 2010

SomeTestCases Wiki page edited by srct114
Revision r44 Edited wiki page SomeTestCases through web user interface.

- Dec 06, 2010

SomeTestCases Wiki page added by srct114
Revision r43 Created wiki page through web user interface.

- Dec 06, 2010

r42 ([No log message]) committed by rct189
[No log message]

- Dec 02, 2010

r41 (List included) committed by andresuribe87
List included

- Dec 02, 2010

r40 ([No log message]) committed by chenlili0603
[No log message]

- Dec 02, 2010

r39 ([No log message]) committed by chenlili0603
[No log message]

- Dec 02, 2010

r38 ([No log message]) committed by rct189
[No log message]

- Dec 02, 2010

r37 (interface) committed by qingfeng...@gmail.com
interface

- Dec 02, 2010

r36 (lib files) committed by qingfeng...@gmail.com
lib files

- Dec 02, 2010

r35 ([No log message]) committed by chenlili0603
[No log message]

- Dec 01, 2010

GraplLib.java (Grapl java lib) file uploaded by qingfeng...@gmail.com

- Dec 01, 2010

  GraplLib.java (Grapl Java Lib) file uploaded by qingfeng...@gmail.com

  10.1.1.9 Earlier this year

- Nov 22, 2010

  r34 (working parser (no conflicts)) committed by rct189
  working parser (no conflicts)

- Nov 18, 2010

  r33 (Andres wants me to write a comment.) committed by rct189
  Andres wants me to write a comment.

- Nov 18, 2010

  r32 ([No log message]) committed by rct189
  [No log message]

- Nov 17, 2010

  r31 (ast, scanner and parser without being built.) committed by andresuribe87
  ast, scanner and parser without being built.

- Nov 15, 2010

  r30 (parser and scanner) committed by andresuribe87
  parser and scanner

- Nov 14, 2010

  r29 ([No log message]) committed by chenlili0603
  [No log message]

- Nov 12, 2010

  r28 ([No log message]) committed by chenlili0603
  [No log message]

- Nov 12, 2010

  Schedule Wiki page edited by rct189
  Revision r27 Edited wiki page Schedule through web user interface.

- Nov 11, 2010

  r26 ([No log message]) committed by chenlili0603
  [No log message]

- Nov 10, 2010

  Home (Wiki homepage of the GRAPL project) Wiki page edited by rct189
  Revision r25 Edited wiki page Home through web user interface.

- Nov 10, 2010

  Home (Wiki homepage of the GRAPL project) Wiki page edited by rct189
  Revision r24 Edited wiki page Home through web user interface.

- Nov 10, 2010

  GRAPL notes.pdf (Handwritten GRAPL notes) file uploaded by rct189

- Nov 10, 2010

  r23 ([No log message]) committed by chenlili0603
  [No log message]

- Nov 09, 2010

  IDE Instruction for Project GRAPL.pdf (IDE INSTRUCTION) file uploaded by qingfeng...@gmail.com

- Nov 05, 2010

  PageName Wiki page deleted by rct189
  Revision r22 Deleting wiki page PageName.

- Nov 05, 2010

  PageName (One-sentence summary of this page.) Wiki page edited by rct189
  Revision r21 Edited wiki page PageName through web user interface.

- Nov 05, 2010

  PageName (One-sentence summary of this page.) Wiki page added by rct189
  Revision r20 Created wiki page through web user interface.

- Nov 05, 2010

  Schedule Wiki page edited by rct189
  Revision r19 Edited wiki page Schedule through web user interface