Glimmer:

An Image Manipulation Language

Terry Tai (netherlight@gmail.com)

Vida Ha (vidaha@gmail.com)

Table of Contents

1 White Paper

1.1 Introduction

Glimmer is a new and exciting programming language for manipulating images and creating special imaging effects, designed to be easy to learn. Extensive programming experience is not a prerequisite for learning Glimmer; instead, we hope even a novice programmer can quickly pick up Glimmer's commands and dive straight into image modifications. To this end, Glimmer natively supports images and pixels as first-order data types, and make it easy to construct advanced filters and effects generators. Glimmer is not designed to be an advanced language or to be used for all general purposes. While Glimmer has limited support for matrix operations, Glimmer is not the best matrix manipulation language around. Instead, this language will allow more advanced control logic for image manipulation than that provided by standard image editing software packages such as Adobe Photoshop.

1.2 Justification

So what is it that Glimmer provides, really?

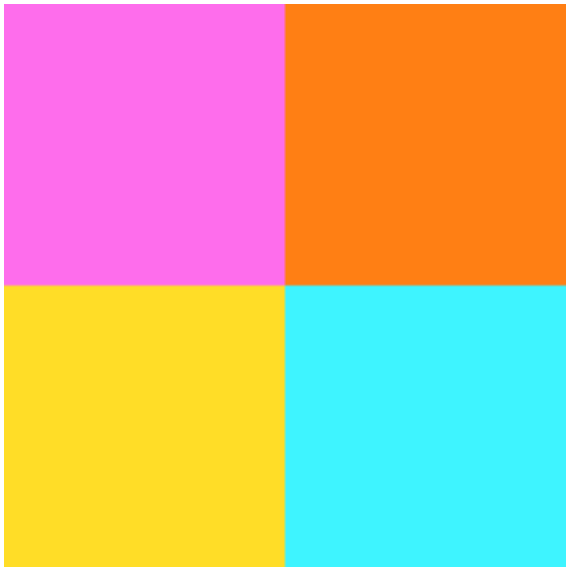
Even very experienced programmers will enjoy the low ramp-up time for learning Glimmer and the *ease* with which someone can edit an image in a programming language setting. While stronger languages such as C can handle the parsing of the bitmap and perform the same numeric operations, the looping constructs needed would be clumsy and prone to error. Glimmer has loop mechanisms specifically tailored for images, and easy ways to access surrounding pixels in an image. With Glimmer, it is possible to do a lot more image manipulation with fewer lines of code.

In addition, Glimmer would be a great language to use to introduce programming to younger students who maybe aren't yet ready to learn a more powerful language. There's no type-checking, objects, or other abstract programming paradigms to learn first. We think it’s intuitive to pick up and that students will appreciate the visual aspect of imagining. A teacher could even use interesting visual patterns to teach elementary algorithms.

Our main hope is that users will have fun using Glimmer for image processing and find interesting applications for this language!

1.3 A Visual Example

Generating gradients is a visually appealing task that many image editors can handle. However, such functions are often limited in their provided options. Glimmer can perform a four-color gradient generation in just over twenty lines of code. This takes the following four colors on the left and generates the gradient on the right. (See the gradient function in utils.glim starting on page 32.)



The four colors used to generate…

a position-proportional gradient.

2 Language Tutorial

In this section we go over a few simple Glimmer programs as a tutorial on learning the Glimmer Language. The LRM ultimately describes each of the commands in detail, for further reference.

2.1 Using Glimmer to Create a Simple Image

This is a first example of a Glimmer program—Glimmer's version of a "Hello, World." Rather than printing out a string—which would just take one line in Glimmer, we create a simple image, set it to be blue everywhere, and save it to a file.

```
// Create a new image that is 100 pixels wide, and 500 pixels in height.
new(rectangle, 100, 500);

// Scan every pixel in the rectangle image and set its color.
scan(p in rectangle) {
  p.pixel_red = 0;     // No red component.
  p.pixel_green = 0;   // No green component.
  p.pixel_blue = 255;  // We like blue!
}

// Save the rectangle image to a file.
save(rectangle, "blue_rectangle.png");
```

2.2 Defining a Simple Function in Glimmer

Here we give a coding snippet for defining a function which sets the color of every pixel in an image.

```
function setImageColor(img, red_val, green_val, blue_val) {
  // Scan every pixel in the image and set its color.
  scan(p in img) {
    p.pixel_red   = red_val;
    p.pixel_green = green_val;
    p.pixel_blue  = blue_val;
  }
}
```

To call the function, we use the expected Java/C-style syntax.

```
setImageColor(my_img, 15, 230, 15);
```

2.3 Two Version of Assignment

Glimmer uses two versions of assignment. The first creates an actual copy of an object if it didn't exist before copies in the core numeric values. The second creates a preference (pointer) to an existing object.

```
load(image, "/home/glimmer/some_image.png");
w = image.image_width;   // This is a COPY.
h ~ image.image_height;  // This is a reference.

w = 1000;  // This will NOT resize the image, since w is a number that just
           // happens to have had the same value as the image width.
h = 2500;  // This will resize the height, since h pointed to the image's
           // height.

save(image, "/home/glimmer/resized_image.png");
```

2.4 A More Complex Glimmer program

```
// Blurs an image by setting each pixel to the average color components
// of the pixels in a surrounding radius.
function Blur(img, radius) {
  // Create a new image that's the averaging area.
  new(window, 2 * radius + 1, 2 * radius + 1);
  scan(p in img) {
    average = [0, 0, 0, 255];
    count = 0;
    scan(local_p in window) {
      diff_x = local_p.pixel_x - radius;  // This'll be the local pixel
      diff_y = local_p.pixel_y - radius;  // references to be used later.
      abs_x = p.pixel_x + diff_x;  // These are the absolute references that
      abs_y = p.pixel_y + diff_y;  // we need to check so we're in bounds.

      // Discard pixels where we're outside the boundary.
      if (abs_x < 0 || abs_x >= img.image_width ||
          abs_y < 0 || abs_y >= img.image_height) {
        continue;
      }

      // Add the three colors into the average pixel.
      average.pixel_red = average.pixel_red + p[diff_x, diff_y].pixel_red;
      average.pixel_green = average.pixel_green +
          p[diff_x, diff_y].pixel_green;
      average.pixel_blue = average.pixel_blue + p[diff_x, diff_y].pixel_blue;
      count = count + 1;
    }

    // Set the average.
    average.pixel_red = average.pixel_red / count;
    average.pixel_green = average.pixel_green / count;
    average.pixel_blue = average.pixel_blue / count;
    p = average;
  }
}

load(fairy, "images/fairy.png");
Blur(fairy, 3);

printstr("Blurring complete.");
```

3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 Comments

Comments in Glimmer mirror the style of C++. They start with two slash characters (//) and end with the first line break.

3.1.2 Tokens

A token is a group of consecutive characters that will be treated by the compiler as an atomic unit.

3.1.3 Whitespace and Line Breaks

Glimmer treats normal spaces (ASCII 0x32), horizontal tabs (0x09), and form feed characters (0x0C) as whitespace. Carriage returns (0x0D) and line feeds (0x0A) are line breaks and are also considered whitespace. However, a carriage return followed immediately by a line feed is considered to be only a single line break (as per DOS/Windows conventions). Whitespace is used as a token separator (as token extraction is greedy) and is otherwise ignored.

3.1.4 Identifiers and Literals

An identifier starts with any single letter (A-Z, a-z) and is followed by zero or more letters or digits or underscores (A-Z, a-z, 0-9, _). Identifiers are case-sensitive and have no maximum length.

Numbers in Glimmer are defined by one or more digits (0-9), optionally followed by a dot (.) and then any number of digits. Negative numbers must have a minus sign (–) preceding. Positive numbers may optionally have a plus sign (+) in front.

While Glimmer doesn't allow strings as a type, string literals are required in specifying filenames and are useful for output. A string is a double-quote (") character followed by any sequence of characters until the first non-escaped double-quote; an escaped double-quote is preceded by a backslash (\"), just like in C.

3.1.5 Keywords

The Glimmer language reserves only a small number of keywords. Identifiers may not be named the following.

- `break`
- `continue`
- `else`

- `function`
- `if`
- `image_height`
- `image_width`
- `in`
- `include`
- `load`
- `new`
- `pixel_alpha`
- `pixel_blue`
- `pixel_green`
- `pixel_red`
- `pixel_x`
- `pixel_y`
- `print`
- `printstr`
- `return`
- `save`
- `scan`

3.1.6 Operators

Operators are special symbols (or sequences of symbols) that specify some action to be performed. Operators in Glimmer include the following:

- `+`
- `-`
- `*`
- `/`
- `%`
- `&&`
- `||`
- `!`
- `>`
- `<`
- `>=`
- `<=`
- `==`
- `!=`
- `=`
- `~`
- `.`
- `( )`
- `[ ]`
- `,`

3.2 Types

Glimmer offers three basic types: images, pixels, and numbers.

3.2.1 Images

Images represent some rectangular block of graphics data and can be created either via the `new` statement or the `load` statement. They can be saved to disk using the `save` statement. The only graphics format that must be supported is the Portable Network Graphics (PNG) format.

Individual pixels may be retrieved from an image using the pixel reference `[x, y]` notation. The returned pixel should be the one corresponding to the one at the x-th column and the y-th row and should be keyed to the current image. `[0, 0]` represents the pixel at the top left corner of the image. The image's dimensions can be found using the `image_height` and `image_width` intrinsics. These two intrinsics should return numbers keyed to the image.

3.2.2 Pixels

A pixel represents the color and transparency values of a single point in an image. Intrinsically, it will be comprised of four numbers: three are devoted to the red, green, and blue color components and the last is for the transparency value.

Pixels can exist in two forms. They can either be free pixels, or they can be keyed to an image. Pixels that are keyed to an image are associated with that image, and changing properties of the pixel will result in corresponding changes in the image.

Declaring a new free pixel uses the syntax:

pixel-declaration : [$expression_1$, $expression_2$, $expression_3$, $expression_4$]

All four expressions must evaluated to numbers. The first will be set to the pixel's red color component. The second is the green; the third is the blue. $expression_4$ is the pixel's alpha transparency value.

We can retrieve the location of a keyed pixel in an image using the `pixel_x` and `pixel_y` intrinsics, both of which should return unkeyed numbers. Colors and transparency values can be obtained using `pixel_red`, `pixel_green`, `pixel_blue`, and `pixel_alpha`, all of which return keyed numbers.

A keyed pixel may be used to reference other pixels in the same image using pixel references `[x, y]`. In this case, however, we should return the pixel (keyed to the same image) using a relative scale; therefore `[x, y]` will refer to the pixel x columns to the right and y rows down from the current pixel.

Allowed values for all four number components encompass the range of Glimmer's number type (i.e., floating-point numbers) if the pixel is free. Otherwise, values should be kept to integers from 0 to 255 as appropriate for images.

3.2.3 Numbers

Numbers serve as the basic real world "number" and should be stored internally in a manner equivalent to Java's `float` type, with its associated precision and features. This differs from many languages in that there is no implicit differentiation between integers and floating-point entities.

Similarly to pixels, numbers may also either by free or keyed. Numbers may be keyed to either an image or a pixel. Numbers keyed to images will represent either the width or the height of the image. Changing the value of such numbers will result in the resizing of the associated image. If a number of keyed to a pixel, it represents either a color component or the alpha value for that pixel. Changing this value will result in the corresponding change in the pixel. Should this pixel itself be keyed to an image, the image will undergo the appropriate color or transparency change as well.

3.3 Identifiers

In Glimmer, identifiers are bound either to an object or to function definitions. A single identifier may be bound to both a single type object and a single function simultaneously. Furthermore, any object of a native type may have multiple identifiers bound to it. In this way, Glimmer operates in a manner similar to Java, where all identifiers reference objects (or functions).

3.3.1 Scopes

Identifiers for objects become available as soon as they are declared. Glimmer provides a top-level scope that begins at the start of the main program statements. Identifiers defined on the top-level will be available until program termination. `if` blocks, `else` blocks, and `scan` blocks will open new, local scopes. Identifiers defined in these blocks will be visible only until the closing of the scope (i.e., until the matching `}`). Furthermore, identifiers in these blocks that share a name with a pre-existing identifier in the parent scope will hide the older identifier until the closing of the current scope.

Functions, on the other hand, all share a single table, regardless of whether the function was declared in the main program or an included file. Because functions are declared before actual program execution and because they cannot be nested, all function definitions should be collected into a single global scope.

3.3.2 Type-Checking

Glimmer is an implicitly typed language. An identifier can take on values of multiple types, though not more than one at once. Calling an intrinsic on an identifier bound to the incorrect type should result in the termination of the program in an error state.

3.4 Program Blocks

A Glimmer program is composed of a single main file and zero or more utility "include" files. Each file is divided into three blocks in the following order:

- Includes – a list of include statements declaring the utility files to be loaded.
- Functions Definitions – a list of function definitions, including the function name, the arguments, and the function body.
- Program Statements – a list of statements that will be executed if the file happens to be the main program.

Execution of the program begins at the first program statement of the main program file.

3.4.1 Include Block

The include block is a list of include statements, consisting of the include keyword followed by a string literal pointing to the desired utility file. Each file thus included is opened and read by the interpreter. All include statements in the new file are recursively processed; functions defined in the new file are added to the global function symbol table. However, any given file is loaded exactly once, even if it is included multiple times, possibly from multiple files. Program statement blocks of included files are not executed.

3.4.2 Function Definition Block

This consists of a list of function declarations and definitions. The syntax for such declarations is as follows:

function-declaration : function *identifier* ((*identifier* (, *identifier*)*)?)
{ (*statement*)* }

The first identifier is the name of the function and how we will call the function in the future. The internal identifiers are a list of names to which the interpreter should alias function arguments when the function is actually invoked. The function body is simply a list of statement to be executed.

3.4.3 Program Statement Block

The program statement block is a list of statements that will be executed. The various types and syntaxes of statements will be discussed in greater detail below.

3.5 Expressions and Operators

Glimmer supports a number of native, intrinsic operations or functions that can be applied to certain base types. Below, we have a list of each along with their associated native type and some descriptions and usage examples. Operators are listed in order of decreasing precedence (items closer to the top are evaluated earlier). All elements within one group share equal priority; associativity is always from left to right.

Type	Intrinsic	Example
any	Parenthesized Expressions	`(a + b)`
any	Function Calls	`GetAverage(i)`
pixel	Attribute Selection (`.pixel_alpha`, `.pixel_red`, `.pixel_green`, `.pixel_blue`)	`p.pixel_red`
image	Attribute Selection (`.image_height`, `.image_width`)	`i.image_height`
pixel	Relative Reference (`[,]`)	`p[1,0]`
image	Absolute Reference (`[,]`)	`i[50,50]`

number	Multiplication, Division, Modulus (`*`, `/`, `%`)	`h * 2`
number	Addition, Subtraction (`+`, `-`)	`r + 1`
number	Comparisons (`>`, `<`, `>=`, `<=`, `==`, `!=`)	`33 >= 4`
number	Logical Negation (`!`)	`!var`
number	Logical And (`&&`)	`var1 && var2`
number	Logical Or (`||`)	`var1 || var2`

3.5.1 Function Calls

Function calls in Glimmer use the following syntax:

function-call : *identifier* ((*expression* (, *expression*)*)?)

When a function call is made, the corresponding identifier is located; if it was not defined previously, the interpreter should terminate execution with an error. A clean scope for identifiers is opened, and the evaluated argument expressions (if any) are bound via aliasing to the function argument identifiers as provided in the function declaration. The function body statements are then executed in order using the new scope. Functions may returns zero or one values to the caller using a `return` statement. Return values are aliases of the original object.

3.5.2 Attribute Selection

Attribute selection is performed by postpending to a pixel or an image a dot (`.`) and the desired attribute. In all cases, this should result in a number keyed to the image or the pixel. The value of the number should represent the selected attribute.

3.5.3 Pixel References

Both absolute (image) and relative (pixel) references are done using the following:

pixel-reference : $expression_1$ [$expression_2$, $expression_3$]

$expression_1$ is the pixel or image on which we desire a reference. The second and third expressions should evaluate to numbers. If $expression_1$ is an image, then we return a pixel keyed to the image corresponding to the pixel in column $expression_2$ and row $expression_3$. If $expression_1$ is a keyed pixel, then we return the pixel shifted right $expression_2$ columns and down $expression_3$ rows. In all other cases, we should error and terminate execution.

If a pixel reference calls for an area outside of an image's boundaries, the result is undefined.

3.5.4 Arithmetic, Multiplicative, and Comparative Operations

These follow standard floating-point procedures, as defined by the Java language specification. Division by 0 results in undefined behavior.

3.5.5 Logical Operations

Since there is no native Boolean type in Glimmer, logical operations are defined slightly differently. True, while not a predefined constant, is taken to be any non-0 value. False is the number 0. Therefore, logical `&&` returns the number 1 if and only if both of its arguments are non-0 and returns 0 in all other cases. Logical `||` returns 0 if and only if both of its arguments are 0, and returns 1 in all other cases. The unary operator `!` returns 0 if its argument was non-0, and 1 if its argument was 0.

3.6 Statements

A statement is a complete command to be executed, and takes the following forms, each of which will be discussed in further detail in the subsequent sections:

statement :
- *set-statement*
- *alias-statement*
- *function-call-statement*
- *return-statement*
- *if-else-statement*
- *scan-statement*
- *continue-statement*
- *break-statement*
- *new-statement*
- *load-statement*
- *save-statement*
- *print-statement*
- *print-string-statement*

3.6.1 Set

As mentioned before, identifiers in Glimmer are references to objects, and thus, one object can have multiple identifiers. The set statement copies the internal values from the lvalue on the right into the expression on the left. An lvalue is an assignable expression, consisting of an identifier followed by zero or more pixel references followed by an optional attribute selector intrinsic.

set-statement : *lvalue* = *expression* ;

Regardless of which type the lvalue evaluates to, the expression on the right of the = must match. If the lvalue happens to be a sole identifier and has not been previously defined, then a new object of the appropriate type is created and the identifier is bound to that object.

In the type of the set statement is a number, the floating-point value for the expression replaces the value of the object referenced by the lvalue. The number's keyed status is unchanged. If it used to be keyed to a pixel or image attribute, it is still keyed as before. This results in actual changes to the pixel and/or the image. On the other hand, if the number was free, it remains so as well, even if the expression on the right was keyed.

If the type of the set statement is a pixel, then using the set statement copies in the value of the four numerical components only. Keyed status is unchanged.

In the case of an image, the image's dimensions are changed to match that of the expression. All pixel values are copied over into the new image. Keyed pixel and number references to the old image are no longer valid and behavior when accessing such objects is undefined.

3.6.2 Alias

alias-statement : *identifier* = *expression* ;

The alias statement operates like the set statement, but copies no values. Instead, we simply link the provided identifier to the object denoted by the expression, creating a new name for the object. Because expressions always return a native type, aliasing and setting identifiers to these objects can result in identical behavior (in particular, when an expression results in an free pixel or number). The following pairs of statement have the same behavior.

```
x = 10;  x ~ 10;
p = [255, 255, 255, 0];  p ~ [255, 255, 255, 0];
```

3.6.3 Function Calls

Function call statements are simple function call expressions where the return value (if any) is discarded. These are probably used mostly for the called function's side effects.

3.6.4 Return

return-statement : return (*expression*)? ;

The return statement is used to return terminate execution of the current block and return some value to the previous control block (if any). If the return statement is given an argument, the expression is evaluated and then the resulting object itself is returned. This statement therefore essentially returns an alias of the argument and not a copy of the object. If a return is used in the main program body and not in the function, this will simply terminate the program's execution; the return argument is evaluated but then ignored.

3.6.5 If-Else

if-else-statement : if (*expression*) { (*statement*)* } (else { (*statement*)* })?

The standard conditional selector in Glimmer consists of the if keyword followed by an expression in parenthesis. This expression must evaluated to a number. If this number is non-0, then the statements inside the first block enclosed by `{` and `}` are executed. If the optional else block is provided, then it will be evaluated if the number expression was 0. Unlike in some languages, the braces are non-optional. Each block, if executed, is provided a new scope that lasts until the exiting of the block.

3.6.6 Scan

scan-statement : scan (*identifier* in *identifier*) { (*statement*)* }

The scan statement first opens a new scope and then aliases the provided first identifier to each pixel in the image referenced by the second identifier. Then, the statement list is run. At the end, the scope is closed and the process repeats, with the pixel now set to the following position. This continues until the statements have been executed on every pixel in the image. Iteration happens in row-major order, going across first and then down.

3.6.7 Continue

continue-statement : continue ;

The continue statement can only be used inside a scan block and terminates execution of the current loop. The scope will be closed and then control will return to the start of the loop, proceeding onto the next pixel.

3.6.8 Break

break-statement : break ;

The break statement can also only be used in a scan loop whose execution will be terminated, but unlike the continue statement, this will not continue iterating with the next pixel. Instead, control will move to the first point after the scan.

3.6.9 Image Input/Output

new-statement : new (*identifier* , *expression* , *expression*) ;
load-statement : load (*identifier* , *string-constant*) ;
save-statement : save (*identifier* , *string-constant*) ;

The new statement creates a new image and binds the identifier to that image. The width is given by the first expression as a number, and the height corresponds to the second. Pixel components are all initialized to 0.

The load statement opens the disk file that is pointed to by the string filename. It must be in Portable Network Graphics format and an image is created with the identifier bound to it.

The save statement takes the identifier that must evaluated to an image and then saves its data to disk in Portable Network Graphics format. The destination file is provided by the string constant.

3.6.10 Print and Print-String

print-statement : print (*expression*) ;
printstr-statement : printstr (*string-constant*) ;

The print statement will evaluate its expression argument and then emit the string representation of the object to standard output. Numbers should use the standard floating-point representation, with the exception that numbers ending in `.0` should have that suffix truncated, allowing integers to appear as such. Pixels should be printed in the format:

pixel-format : (Image(*x*,*y*):)? (*red*,*green*,*blue*)/*alpha*

with the optional image section displayed only for bound images. All values should follow the standard number format. Images should be printed as follows:

image-format : Image: *height* rows by *width* columns

where height and width display using the number formatting rules.

The printstr statement will simply echo the given string to standard output.

Both printing statement should appending a trailing newline.

4 Project Plan

4.1 Team Responsibilities

This was the planned separation of team responsibilities. We did not feel it was necessary with a two person team to elect a team leader which, although unadvised, worked out for us.

Component	Primary Person in Charge
Lexer	Terry Tai
Parser	Vida Ha
Symbol Table and Glimmer Type Implementations	Terry Tai
Walker/Interpreter	Vida Ha
Testing	Terry Tai
Final Project Report	Vida Ha

4.2 Planned Project Timeline

There were dates we set for crucial milestones.

Milestone	Deadline
Proposal	6/4/2008
LRM and Grammar Acceptance	6/20/2008
AST Generation Complete	6/30/2008
Symbol Table and Glimmer Type Implementations	7/14/2008
Complete Walk of AST	7/14/2008
Integration of Walker and Glimmer Libraries	7/25/2008
Finalize Testing and Debugging	8/1/2008
Final Report	8/11/2008

4.3 Project Log

This table shows a log of when crucial milestones where actually met and the main person who completed that task. In reality, tasks where much more of a joint effort than this table might suggest—a typical task was commenced by one person, but the tricky issues were ironed out together.

Task	Date Completed	Name
Brainstorm ideas & draft proposal on favorites	6/1/2008	TT, VH
Agree on project	6/2/2008	TT, VH
Proposal writeup	6/4/2008	TT, VH
Familiarize with ANTLR, get main functions to print out ASTs, draft some testcases for Lexer and Parser.	6/9/2008	TT, VH

Lexer complete	6/13/2008	TT
Parser accepting grammar + some work on AST generation	6/18/2008	VH
LRM writeup	6/19/2008	TT
Parser AST generation complete, tests on AST generation.	6/27/2008	TT
Walker walks AST and prints out nodes visited, minor grammar changes required.	7/18/2008	VH
Symbol Table and Java libraries complete with unittesting.	7/18/2008	TT
Integration of Java libraries into walker code.	7/31/2008	VH
Additional modifications to Java libraries as needed.	7/31/2008	TT
Integration testing and fully implement a few more language features.	8/4/2008	TT
LRM Rewrite	8/10/2008	TT
Final Report Write up	8/10/2008	VH

4.4 Development Environment

The code was developed on UNIX machines with Perforce as the version control software. Tests were run on a nightly cron job, with both team members emailed in event of test breakage. The Lexer, Parser, and Walker code was done in Antler 2.x and the Glimmer libraries were implemented in Java 1.5.

4.5 Code Style Conventions

4.5.1 Antlr Code Style Conventions

- Uniform spacing and indentation.
- 80 character line limit when possible.
- Logical and uniform names for variables – i.e. all statements are named <xyz-statement>
- Write code in a logical ordering – such as the statements to walk the AST are defined in the same order as the statements we used to generate the AST.
- Comment for tricky code blocks to explain behavior.

4.5.2 Java Code Style Conventions

For Java code, we decided to adopt the official code conventions endorsed by Sun, and detailed on this webpage: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html.

5 Architectural Design

5.1 Overview

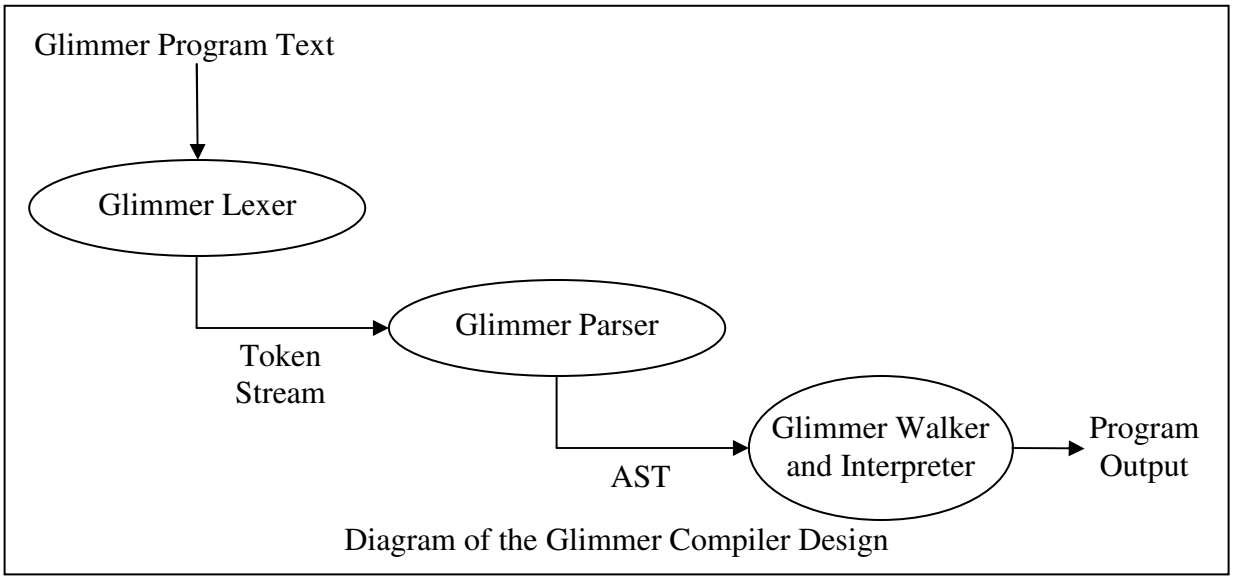


Diagram of the Glimmer Compiler Design

These are the stages of the Glimmer Compiler, as depicted in the diagram above.

1. The complete Glimmer program text is input into the Glimmer Lexer which produces a stream of tokens.
2. The token stream is input to the Glimmer parser, producing an abstract syntax tree (AST).
3. The AST is input into the Glimmer walker/interpreter which walks the AST and interprets it, executing the code immediately.

5.2 Glimmer Lexer

The GlimmerLexer processes the program text and produces tokens. It removes comments, extra white spaces, and decides what characters form a logic token. Anytime an illegal token is encountered, the GlimmerLexer throws a TokenStreamException.

Here is a coding snippet to show how to create a GlimmerLexer and pass it into the Glimmer Parser:

```
GlimmerLexer lexer = new GlimmerLexer(programText);
GlimmerParser parser = new GlimmerParser(lexer);
```

Terry was responsible for implementation of this component.

5.3 Glimmer Parser

The GlimmerParser processes the TokenStream from the GlimmerLexer, catching and reporting any TokenExceptions from the Glimmer lexer. The Glimmer parser will also throw and catch recognition exceptions if it sees illegal sequences of tokens. The GlimmerParser will continue to try to parse a program even after an exception has occurred, but has an internal variable, parserError, which will be set to true in event of an exception. While parsing, the GlimmerParser builds the abstract syntax tree for the program. The walker should check that the parserError variable is not true before asking for the AST.

This is a coding snippet of how to use the Glimmer Parser:

```
GlimmerParser parser = new GlimmerParser(lexer);
parser.program();
if (parser.hasParserError()) {
throw new RecognitionException(
  "Will not run with parse errors.");
}
CommonAST parseTree (CommonAST)parser.getAST();
```

Vida was responsible for implementation of this component to get the grammar accepting, and Terry worked on the AST generation.

5.4 GlimmerType Libraries and SymbolTable

The Glimmer compiler includes java implementations for internal Glimmer types, as well as a SymbolTable and a special GlimmerException type. The following are a list of Glimmer classes implemented in Java:

- GlimmerException
- GlimmerSymbolTable
- GlimmerType – The interface that all GlimmerTypes must define.
- GlimmerID
- GlimmerNum
- GlimmerPixel
- GlimmerImage
- GlimmerFunction

In particular, the GlimmerType interface is interesting, because it has function names which intuitively follow the AST components which the walker will see. Please see Section 8 for the exact interface.

Terry was responsible for implementing these classes.

5.5 Glimmer Walker/Interpreter

The GlimmerWalker walks the AST and calls the Glimmer Java library functions to execute the program code. This is an example of how `set` is handled by the walker:

```
| #(SET first=expr second=expr)
  {
    // Check that the rvalue is valid.
    if (second == null || !second.validRValue()) {
      throw new GlimmerException(
          "Invalid or undefined variable: " +
          (second != null ? second.toString() : "<null>"));
    }

    // Now set it if appropriate.
    if (first instanceof GlimmerID) {
      // Either create a new instance...
      current_table.add(((GlimmerID)first).getID(),
                        second.copy());
    } else {
      // Otherwise, just mutate the current object.
      first.replace(second);
    }
  }
```

Please see Section 8.3 for more details on the walker.

Vida was responsible for the walking the AST and the integration of the Java libraries into the walker code.

6 Test Plan

The plan for testing the Glimmer Compiler was to first test the Lexer/Parser's AST generation, unittest the Glimmer Java library classes, and finally - full integration testing.

6.1 Testing the Lexer/Parser

The testing plan for the lexer and parser was to create sample programs and make sure the they accepted these programs and produced the correct AST for them. The input was .glim files and the expected output was .golden files with a text representation of the expected AST. Vida started some of the .glim files for testing that the grammars were accepting the right thing; Terry cleaned these tests up and added the components to test AST generation.

6.1.1 Testing Assignments

assigments.glim

```
function Return2() {
  return 2;
}

one = 1;
two = 2;
three = one + two;
four = 1 + two * three / (1 + 1);

load(img, "/tmp/image.bmp");

p1 = img[50,50];
p2 = p1[-5,10];
p1.pixel_red = four;
img.image_height = img.image_width * Return2();
```

assignments.golden

```
( PROGRAM INCLUDE_LIST ( FUNC_LIST ( function Return2 FUNC_ARGS ( STMT_LIST (
return 2 ) ) ) ) ( STMT_LIST ( = one 1 ) ( = two 2 ) ( = three ( + one two )
) ( = four ( + 1 ( / ( * two three ) ( + 1 1 ) ) ) ) ( load img
/tmp/image.bmp ) ( = p1 ( [ img 50 50 ) ) ( = p2 ( [ p1 ( UMINUS 5 ) 10 ) ) (
= ( pixel_red p1 ) four ) ( = ( image_height img ) ( * ( image_width img ) (
FUNC_CALL Return2 FUNC_ARGS ) ) ) ) )
```

6.1.2 Testing Image Intrinsics

image-intrinsics.glim

```
load(image, "/tmp/image.bmp");
save(image, "/tmp/image.bmp");
new(image, 100, 200);
```

image-intrinsics.golden

```
( PROGRAM INCLUDE_LIST FUNC_LIST ( STMT_LIST ( load image /tmp/image.bmp ) (
save image /tmp/image.bmp ) ( new image 100 200 ) ) )
```

6.1.3 Testing Functions

functions.glim

```
// Tests the function declarations and syntax for Glimmer.
function Func00() {
  return;
}

function Func10(param) {
  return;
}

function Func20(param1, param2) {
  return;
}

function Func01() {
  retval = 10;
  return retval;
}

function Func11(param) {
  retval = param + param;
  return retval;
}

// Main Program
Func00();
Func10(3);
Func20(3, 4);
a = Func01();
a = Func11(3);
```

functions.golden

```
( PROGRAM INCLUDE_LIST ( FUNC_LIST ( function Func00 FUNC_ARGS ( STMT_LIST
return ) ) ( function Func10 ( FUNC_ARGS param ) ( STMT_LIST return ) ) (
function Func20 ( FUNC_ARGS param1 param2 ) ( STMT_LIST return ) ) ( function
Func01 FUNC_ARGS ( STMT_LIST ( = retval 10 ) ( return retval ) ) ) ( function
Func11 ( FUNC_ARGS param ) ( STMT_LIST ( = retval ( + param param ) ) (
return retval ) ) ) ) ( STMT_LIST ( FUNC_CALL Func00 FUNC_ARGS ) ( FUNC_CALL
Func10 ( FUNC_ARGS 3 ) ) ( FUNC_CALL Func20 ( FUNC_ARGS 3 4 ) ) ( = a (
FUNC_CALL Func01 FUNC_ARGS ) ) ( = a ( FUNC_CALL Func11 ( FUNC_ARGS 3 ) ) ) )
)
```

6.1.4 Full test 1

full.glim

```
include "parsertests/full2.glim"

function ReturnOne() {
  return 1;
}

function Square(xx) {
```

```
  return xx * xx;
}

function Func(image) {
  image[5,5] = [1, 2, 3, 4];
}

function Func2(num, image) {
  image.image_height = num;
}

five = 5;
load(i, "my_test_image.bmp");
p = [0, 198, 198, 198];
q ~ p;

Func(image);

scan(p in i) {
  ten = 10;
  p[-3,4].pixel_red = 1;
  p[1,ReturnOne()][1,1].pixel_green = 2;
}

s[1,1][2,2][3,3][4,4] = 2 + 3 * p[-1,-1].pixel_red / 5 || !1;

if (1) {
  number = 5 + 4 && Func2(s + -0.3, Square(s));
  xx = ReturnOne();
} else {
  number = 1;
}
```

full.golden

```
( PROGRAM ( INCLUDE_LIST ( include parsertests/full2.glim ) ) ( FUNC_LIST (
function ReturnOne FUNC_ARGS ( STMT_LIST ( return 1 ) ) ) ( function Square (
FUNC_ARGS xx ) ( STMT_LIST ( return ( * xx xx ) ) ) ) ( function Func (
FUNC_ARGS image ) ( STMT_LIST ( = ( [ image 5 5 ) ( PIXEL_CREATE 1 2 3 4 ) )
) ) ( function Func2 ( FUNC_ARGS num image ) ( STMT_LIST ( = ( image_height
image ) num ) ) ) ) ( STMT_LIST ( = five 5 ) ( load i my_test_image.bmp ) ( =
p ( PIXEL_CREATE 0 198 198 198 ) ) ( ~ q p ) ( FUNC_CALL Func ( FUNC_ARGS
image ) ) ( scan p i ( STMT_LIST ( = ten 10 ) ( = ( pixel_red ( [ p ( UMINUS
3 ) 4 ) ) 1 ) ( = ( pixel_green ( [ ( [ p 1 ( FUNC_CALL ReturnOne FUNC_ARGS )
) 1 1 ) ) 2 ) ) ) ( = ( [ ( [ ( [ ( [ ( [ s 1 1 ) 2 2 ) 3 3 ) 4 4 ) ( || ( + 2 (
/ ( * 3 ( pixel_red ( [ p ( UMINUS 1 ) ( UMINUS 1 ) ) ) ) 5 ) ) ( ! 1 ) ) ) (
if 1 ( STMT_LIST ( = number ( && ( + 5 4 ) ( FUNC_CALL Func2 ( FUNC_ARGS ( +
s ( UMINUS 0.3 ) ) ( FUNC_CALL Square ( FUNC_ARGS s ) ) ) ) ) ) ( = xx (
FUNC_CALL ReturnOne FUNC_ARGS ) ) ( STMT_LIST ( = number 1 ) ) ) ) )
```

6.1.5 Full Test 2

full2.glim

```
function make_red_pixels_white(image) {
  return;
}
```

```
function make_red_pixels_white(image, image2, image3) {
  return make_red_pixels_white(image2);
}

five = 5;
load(i, "my_test_image.bmp");
load(i2, "my_test_image.bmp");
p = [0, 198, 198, 198];

s = 1 + 2 + 3 + 4 + 5;
pix.pixel_red = 10;
img[1,2][3,4][5,6].pixel_red = 10;

make_red_pixels_white();
make_red_pixels_white(image);
make_red_pixels_white(image, image2, image3);

scan(p in i) {
  ten = 10;
  p[-3,4].pixel_red = 1;
  p[1,return_one()][1,1] = 2;
}

if (bool) {
  number = 5 + 4 && make_greeen(sjsjsj + -0.3, func2(asdf));
} else {
  number = 1;
}
```

full2.golden

```
( PROGRAM INCLUDE_LIST ( FUNC_LIST ( function make_red_pixels_white (
FUNC_ARGS image ) ( STMT_LIST return ) ) ( function make_red_pixels_white (
FUNC_ARGS image image2 image3 ) ( STMT_LIST ( return ( FUNC_CALL
make_red_pixels_white ( FUNC_ARGS image2 ) ) ) ) ) ( STMT_LIST ( = five 5 )
( load i my_test_image.bmp ) ( load i2 my_test_image.bmp ) ( = p (
PIXEL_CREATE 0 198 198 198 ) ) ( = s ( + ( + ( + ( + 1 2 ) 3 ) 4 ) 5 ) ) ( =
( pixel_red pix ) 10 ) ( = ( pixel_red ( [ ( [ ( [ img 1 2 ) 3 4 ) 5 6 ) ) 10
) ( FUNC_CALL make_red_pixels_white FUNC_ARGS ) ( FUNC_CALL
make_red_pixels_white ( FUNC_ARGS image ) ) ( FUNC_CALL make_red_pixels_white
( FUNC_ARGS image image2 image3 ) ) ( scan p i ( STMT_LIST ( = ten 10 ) ( = (
pixel_red ( [ p ( UMINUS 3 ) 4 ) ) 1 ) ( = ( [ ( [ p 1 ( FUNC_CALL return_one
FUNC_ARGS ) ) 1 1 ) 2 ) ) ) ( if bool ( STMT_LIST ( = number ( && ( + 5 4 ) (
FUNC_CALL make_greeen ( FUNC_ARGS ( + sjsjsj ( UMINUS 0.3 ) ) ( FUNC_CALL
func2 ( FUNC_ARGS asdf ) ) ) ) ) ) ) ( STMT_LIST ( = number 1 ) ) ) ) )
```

6.2 Testing the Java Libraries

These tests were written by Terry Tai. They were designed to test the Java functionality of the language.

6.2.1 Symbol Table Testing

SymbolTableTest.java

```
/**
```

```
 * Test the symbol table.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class SymbolTableTest {
  static public void main(String[] args) {
    try {
      GlimmerType n1 = new GlimmerNum(1.0f);
      GlimmerType n2 = new GlimmerNum(2.0f);
      GlimmerType p1 = new GlimmerPixel(1, 2, 3, 4);
      GlimmerType i1 = new GlimmerImage("../images/color_wheel.png");

      GlimmerSymbolTable t1 = new GlimmerSymbolTable();
      t1.add("n1", n1);
      t1.add("n2", n2);
      t1.add("p1", p1);
      t1.add("i1", i1);

      // Basic Retrieval
      System.out.println("Retrieving t1/n1: " + (n1 == t1.get("n1")));
      System.out.println("Retrieving t1/n2: " + (n2 == t1.get("n2")));
      System.out.println("Retrieving t1/p1: " + (p1 == t1.get("p1")));
      System.out.println("Retrieving t1/i1: " + (i1 == t1.get("i1")));

      // Create a new scope with a shadowing name.
      GlimmerSymbolTable t2 = t1.openScope();
      GlimmerType t2n1 = new GlimmerNum(3.0f);
      t2.add("n1", t2n1);
      System.out.println("Retrieving t2/n1: " + (t2n1 == t2.get("n1")));
      // This should still work.
      System.out.println("Retrieving t1/n1: " + (n1 == t1.get("n1")));
      System.out.println("Retrieving t2/n2: " + (n2 == t2.get("n2")));
      System.out.println("Retrieving t2/p1: " + (p1 == t2.get("p1")));
      System.out.println("Retrieving t2/i1: " + (i1 == t2.get("i1")));

      // Check that afterwards, the parent still functions correctly.
      GlimmerSymbolTable t1_2 = t2.getParent();
      System.out.println("Retrieving t1_2/n1: " + (n1 == t1_2.get("n1")));
      System.out.println("Retrieving t1_2/n2: " + (n2 == t1_2.get("n2")));
      System.out.println("Retrieving t1_2/p1: " + (p1 == t1_2.get("p1")));
      System.out.println("Retrieving t1_2/i1: " + (i1 == t1_2.get("i1")));

      System.out.println(t1.toString());
    } catch (Exception e) {
      System.err.println("Error: " + e.getMessage());
      System.out.println("Error: " + e.getMessage());
    }
  }
}
```

SymbolTableTest.golden

```
Retrieving t1/n1: true
Retrieving t1/n2: true
Retrieving t1/p1: true
Retrieving t1/i1: true
Retrieving t2/n1: true
```

```
Retrieving t1/n1: true
Retrieving t2/n2: true
Retrieving t2/p1: true
Retrieving t2/i1: true
Retrieving t1_2/n1: true
Retrieving t1_2/n2: true
Retrieving t1_2/p1: true
Retrieving t1_2/i1: true
Symbol Table:
n2: 2
n1: 1
i1: Image: 326 rows by 400 columns
p1: (1,2,3)/4
```

6.2.2 Glimmer Type Testing

TypesTest.java

```
/**
 * Test the types.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class TypesTest {
  static public void main(String[] args) {
    try {
      GlimmerType ten = new GlimmerNum(10.0f);
      GlimmerType two = new GlimmerNum(2.0f);
      System.out.println("Type: " + ten.typeName());
      System.out.println("Value: " + ten);
      System.out.println("10 + 2 = " + (ten.add(two)));
      System.out.println("10 - 2 = " + (ten.sub(two)));
      System.out.println("10 * 2 = " + (ten.mul(two)));
      System.out.println("10 / 2 = " + (ten.div(two)));
      System.out.println("-10 = " + ten.neg());
      System.out.println("!10 = " + ten.not());
      System.out.println("10 < 2 = " + ten.lt(two));
      System.out.println("10 > 2 = " + ten.gt(two));
      System.out.println("10 <= 2 = " + ten.le(two));
      System.out.println("2 >= 2 = " + two.ge(two));
      System.out.println("2 == 2 = " + two.eq(two));
      System.out.println("2 != 2 = " + two.ne(two));
      System.out.println("2 is true: " + ((GlimmerNum)two).isTrue());
      System.out.println("false is true: " +
                         GlimmerNum.FALSE_VALUE.isTrue());
      System.out.println("true is true: " + GlimmerNum.TRUE_VALUE.isTrue());

      GlimmerType i = new GlimmerImage("../images/color_wheel.png");
      System.out.println("Type: " + i.typeName());
      System.out.println("Filename: " + i);
      System.out.println("Height: " + i.imageHeight());
      System.out.println("Width: " + i.imageWidth());

      GlimmerType p = i.pixelRef(139, 67);
      System.out.println("Type: " + p.typeName());
      System.out.println("Pixel Values: " + p);
      GlimmerType p2 = p.pixelRef(114, 25);
```

```
      System.out.println("Pixel Values: " + p2);

      System.out.println("Red: " + p2.pixelRed());
      System.out.println("Green: " + p2.pixelGreen());
      System.out.println("Blue: " + p2.pixelBlue());
      System.out.println("Alpha: " + p2.pixelAlpha());
      System.out.println("X: " + p2.pixelX());
      System.out.println("Y: " + p2.pixelY());

      GlimmerType newi = new GlimmerImage();
      ((GlimmerImage)newi).intrinsicNew(10, 20);
      System.out.println("Height: " + newi.imageHeight());
      System.out.println("Width: " + newi.imageWidth());
      ((GlimmerImage)newi).intrinsicLoad("../images/color_wheel.png");
      ((GlimmerImage)newi).intrinsicWrite("/tmp/copy.png");

      String[] funcargs = {"image", "color", "row", "column"};
      GlimmerFunction func = new GlimmerFunction(funcargs, null);
      System.out.println(func);

      GlimmerType t = new GlimmerType();
      System.out.println("Type: " + t.typeName());

    } catch (Exception e) {
      System.err.println("Error: " + e.getMessage());
    }
  }
}
```

TypesTest.golden

```
Type: Num
Value: 10
10 + 2 = 12
10 - 2 = 8
10 * 2 = 20
10 / 2 = 5
-10 = -10
!10 = 0
10 < 2 = 0
10 > 2 = 1
10 <= 2 = 0
2 >= 2 = 1
2 == 2 = 1
2 != 2 = 0
2 is true: true
false is true: false
true is true: true
Type: Image
Filename: Image: 326 rows by 400 columns
Height: 326
Width: 400
Type: Pixel
Pixel Values: Image(139,67):(0,255,0)/255
Pixel Values: Image(253,92):(251,244,65)/255
Red: 251
Green: 244
```

```
Blue: 65
```

6.2.3 Image Manipulation Library Testing

ImageManipulation.java

```
import java.lang.Math;
import java.io.FileInputStream;

/**
 * Test the types.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class ImageManipulation {
  static public void main(String[] args) {
    String old_filename = "../images/fairy.png";
    String gold_filename = "../images/golden_fairy.png";
    String new_filename = "/tmp/golden_fairy.png";

    try {
      GlimmerImage img = new GlimmerImage(old_filename);
      System.out.println("Type: " + img.typeName());
      System.out.println("Filename: " + img);
      System.out.println("Height: " + img.imageHeight());
      System.out.println("Width: " + img.imageWidth());

      GlimmerPixel white = new GlimmerPixel(255, 255, 255, 255);

      GlimmerPixel pix = (GlimmerPixel)img.pixelRef(0, 0);
      do {
        // Rotate colors.
        GlimmerPixel newpix = new GlimmerPixel(
            (int)pix.pixelGreen().getValue(),
            (int)pix.pixelRed().getValue(),
            (int)pix.pixelBlue().getValue(),
            (int)pix.pixelAlpha().getValue());
        pix.replace(newpix);

        // If white, then...
        if (((GlimmerNum)pix.eq(white)).getValue() == 1.0) {
          // Do nothing - this shouldn't crash.
        }

        // Use two methods to make it a little more red and blue.
        pix.pixelRed().replace(new GlimmerNum(
            Math.min(255.0f, pix.pixelRed().getValue() + 40)));
        pix.changeComponent(new GlimmerNum(
            Math.min(255.0f, pix.pixelBlue().getValue() + 25)),
            NumberField.PIXEL_BLUE);
      } while ((pix = (GlimmerPixel)pix.getNext()) != null);

      img.intrinsicWrite(new_filename);

      // Perform the difference.
      FileInputStream gold_file = new FileInputStream(gold_filename);
      FileInputStream new_file = new FileInputStream(new_filename);
```

```
      while (true) {
        int gold_num = gold_file.read();
        int new_num = new_file.read();
        if (gold_num != new_num) {
          System.out.println("Differences found in images: " +
                             gold_filename + " vs. " + new_filename);
          break;
        }
        if (gold_num == -1)
          break;
      }

      GlimmerNum white_red = white.pixelRed();
      white_red.replace(new GlimmerNum(5.0f));
      System.out.println("New White: " + white_red);
    } catch (Exception e) {
      System.err.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

ImageManipulation.golden

```
Type: Image
Filename: Image: 375 rows by 285 columns
Height: 375
Width: 285
New White: 5
```

6.3 Full Integration Testing

These tests were written by Terry Tai.

6.3.1 utils.glim

Utils.glim is a glimmer utility used for the full test suites and contained a variety of useful functions.

```
include "fulltests/utils.glim"  // This should not crash (or loop
infinitely)!

function CheckImageEquals(img1, img2) {
  if (img1.image_height != img2.image_height ||
      img1.image_width != img2.image_width) {
    return 0;
  }
  scan (p in img1) {
    if (p != img2[p.pixel_x, p.pixel_y]) {
      return 0;
    }
  }
  return 1;
}
```

```
function ReturnMod4(num) {
  // This is a repeat function, but since it's the same, we should be okay.
  return num % 4;
}

// Blurs an image by setting each pixel to the average color components
// of the pixels in a surrounding radius.
function Blur(img, radius) {
  new(window, 2 * radius + 1, 2 * radius + 1);
  scan(p in img) {
    average = [0, 0, 0, 255];
    count = 0;
    scan(local_p in window) {
      diff_x = local_p.pixel_x - radius;
      diff_y = local_p.pixel_y - radius;
      abs_x = p.pixel_x + diff_x;
      abs_y = p.pixel_y + diff_y;

      if (abs_x < 0 || abs_x >= img.image_width ||
          abs_y < 0 || abs_y >= img.image_height) {
      continue;
      }

      average.pixel_red = average.pixel_red + p[diff_x, diff_y].pixel_red;
      average.pixel_green = average.pixel_green + p[diff_x,
          diff_y].pixel_green;
      average.pixel_blue = average.pixel_blue + p[diff_x, diff_y].pixel_blue;
      count = count + 1;
    }

    average.pixel_red = average.pixel_red / count;
    average.pixel_green = average.pixel_green / count;
    average.pixel_blue = average.pixel_blue / count;

    p = average;
  }
}

// Generates a gradient using two pixel colors.
function Gradient(left_color, right_color, img) {
  Gradient4(left_color, right_color, left_color, right_color, img);
}

// Generates a gradient using 4 colors, one for each corner.
function Gradient4(tl_color, tr_color, bl_color, br_color, img) {
  new(top, img.image_width, 1);
  new(bottom, img.image_width, 1);

  scan(p in top) {
    ratio = p.pixel_x / (top.image_width - 1);
    p.pixel_red = tl_color.pixel_red + (tr_color.pixel_red -
                                        tl_color.pixel_red) * ratio;
    p.pixel_green = tl_color.pixel_green + (tr_color.pixel_green -
                                            tl_color.pixel_green) * ratio;
    p.pixel_blue = tl_color.pixel_blue + (tr_color.pixel_blue -
                                          tl_color.pixel_blue) * ratio;
  }
```

```
  scan(p in bottom) {
    ratio = p.pixel_x / (bottom.image_width - 1);
    p.pixel_red = bl_color.pixel_red + (br_color.pixel_red -
                                        bl_color.pixel_red) * ratio;
    p.pixel_green = bl_color.pixel_green + (br_color.pixel_green -
                                           bl_color.pixel_green) * ratio;
    p.pixel_blue = bl_color.pixel_blue + (br_color.pixel_blue -
                                          bl_color.pixel_blue) * ratio;
  }

  scan(p in img) {
    ratio = p.pixel_y / (img.image_height - 1);
    p.pixel_red = top[p.pixel_x, 0].pixel_red +
                  ratio * (bottom[p.pixel_x, 0].pixel_red -
                  top[p.pixel_x, 0].pixel_red);
    p.pixel_green = top[p.pixel_x, 0].pixel_green +
                    ratio * (bottom[p.pixel_x, 0].pixel_green -
                    top[p.pixel_x, 0].pixel_green);
    p.pixel_blue = top[p.pixel_x, 0].pixel_blue +
                   ratio * (bottom[p.pixel_x, 0].pixel_blue -
                   top[p.pixel_x, 0].pixel_blue);
    p.pixel_alpha = 255;
  }
}

a = 10;     // These statements shouldn't run, unless we actually run
utils.glim.
print(a);
print(a);
print(a);
print(a);
print(a);
print(a);
print(a);
print(a);
print(a);
print(a);
```

6.3.2 Test 1

test1.glim

```
include "fulltests/utils.glim"

function ReturnMod4(num) {
  return num % 4;
}

function Square(xx) {
  return xx * xx;
  a = b + c; // This should never be seen!
}

function MakeBright(n) {
  n = 255;
}
```

```
five = 5;
p = [18, 7, 2, 1];
q ~ p;

print(five);
print(5);
print(p);
print(Square(q.pixel_green));
q.pixel_green = ReturnMod4(6.5) * 40;  // Should be 100.
print(Square(p.pixel_green));

load(img, "images/fairy.png");
print(img.image_height);
print(img.image_width);

// Image sizes = 285 (width) x 375 (height)
new(img2, img.image_width, img.image_height);

scan(p in img) {
//  img2[p.pixelX, p.pixelY] = [1, 2, 3, 4];
  img2[p.pixel_x, p.pixel_y] = p;
}

scan(p in img2) {
  if (p.pixel_y > img2.image_height / 2) {
    break;
  }

  MakeBright(p.pixel_blue);

  if (p.pixel_x > img2.image_width / 2) {
    continue;
  } else {
    MakeBright(p.pixel_red);
  }
}
printstr("Done brightening.");

black = [0, 0, 0, 255];
scan (p in img2) {
  if (p.pixel_y > 0 && p.pixel_y < 3) {
    p[0, -1] = black;
  }
  if (p.pixel_y < img2.image_height - 1 &&
      p.pixel_y > img2.image_height - 4) {
    p[0, 1] = black;
  }

  if (p.pixel_x > 0 && p.pixel_x < 3) {
    p[-1, 0] = [255, 128, 0, 255];
  }
  if (p.pixel_x < img2.image_width - 1 && p.pixel_x > img2.image_width - 4) {
    p[1, 0] = [0, 255, 0, 255];
  }
}

// Adds a dot at (10, 10).  If we started with p = img2[10, 10] then this
```

```
// wouldn't work.
p ~ img2[10, 10];
q ~ p;
print(q);
p = [0, 0, 0, 255];
print(q);

r = p.pixel_red;
g ~ p.pixel_green;
g2 ~ g;
r = 255;    // Should do nothing.
g = 255;

print(p);
print(g2);
print(img2[10, 10]);

printstr("Done adding border and dot.");
save(img2, "/tmp/border.png");

img3 = img2;
w = img3.image_width;
h ~ img3.image_height;

print(w);
print(h);

w = 200;                  // This shouldn't change the width since we used a
set.
h = 300;                  // This will, however, crop off her feet :(.
img4 ~ img3;              // Copy img3 in name only.
img4.image_height = 450;  // This restores it, but it'll be white now.

print(w);
print(h);
print(img3.image_width);
print(img3.image_height);

save(img3, "/tmp/enlarged.png");

load(check_border_img, "images/border_fairy.png");
load(test_border_img, "/tmp/border.png");
load(check_enlarged_img, "images/enlarged_fairy.png");
load(test_enlarged_img, "/tmp/enlarged.png");
print(CheckImageEquals(check_border_img, test_border_img));
print(CheckImageEquals(check_enlarged_img, test_enlarged_img));
```

test1.golden

```
Skipping repeat inclusion of fulltests/utils.glim
5
5
(18,7,2)/1
49
10000
375
285
```

```
Done brightening.
Image(10,10):(255,255,255)/255
Image(10,10):(0,0,0)/255
Image(10,10):(0,255,0)/255
255
Image(10,10):(0,255,0)/255
Done adding border and dot.
285
375
200
450
285
450
1
1
```

6.3.3 Test 2

test2.glim

```
include "fulltests/utils.glim"

load(fairy, "images/fairy.png");
Blur(fairy, 3);

printstr("Blurring fairy complete.");

new(grad, 400, 400);
pink = [254, 109, 236, 0];
orange = [255, 127, 20, 0];
yellow = [255, 221, 39, 0];
blue = [62, 244, 255, 0];
Gradient4(pink, orange, yellow, blue, grad);

printstr("Gradient generation complete.");

load(test_blurry_fairy, "images/blurry_fairy.png");
load(test_grad, "images/gradient.png");
print(CheckImageEquals(fairy, test_blurry_fairy));
print(CheckImageEquals(grad, test_grad));
```

test2.golden

```
Skipping repeat inclusion of fulltests/utils.glim
Blurring fairy complete.
Gradient generation complete.
1
1
```

6.3.4 Test Control

testcontrol.glim

```
include "fulltests/utils.glim"

function Two() {
  printstr("aaaaaaaaaaaaaaaaaaaaa");
```

```
  k = One();
  printstr("bbbbbbbbbbbbbbbbbbbbbb");
  return k;
}

function One() {
  printstr("one-1");
  printstr("one-2");
  new(img, 1, 10);
  scan(p in img) {
    print(p.pixel_y);
    if (p.pixel_y >= 7) {
      printstr("break...");
      break;
    }
    if (p.pixel_y >= 4) {
      printstr("continue...");
      continue;
    }
    print(p.pixel_y * p.pixel_y);
  }

  i = 0;
  j = i;
  scan(p in img) {
    scan(q in img) {
      i = i + 1;
      if (p.pixel_y >= 5) {
        return i;  // Should be 5 * 10 + 1 = 51
      }
      j = j + 1;
    }
  }

  print(j);
  return i;
}

i = 10;
j = i;
i = i + 1;
print(i);
print(j);

print(Two());
printstr("done");
```

testcontrol.golden

```
Skipping repeat inclusion of fulltests/utils.glim
11
10
aaaaaaaaaaaaaaaaaaaaa
one-1
one-2
0
0
```

```
1
1
2
4
3
9
4
continue...
5
continue...
6
continue...
7
break...
bbbbbbbbbbbbbbbbbbbbb
51
done
```

7 Lessons Learned / Advice for Future Groups

7.1.1 Have patience with the parser

It was difficult to get started with the parser, because not only did that require understanding of ANTLR syntax, but it also meant that you had to fit the whole language in your mind at once for the first time. Up until that point, we mostly just had an idea of little language features we wanted to include. Deciding how it was all going to fit together was a hard hurdle to overcome, but once it did come together, adding additional things was relatively simple.

7.1.2 The AST orders things differently than the language syntax

The most difficult bug we encountered with was building the AST's to handle the attribute selection intrinsics, and the pixel reference. Since our syntax has those functions come after the thing they are called on, we both got stuck on thinking that our AST should look that way too and spent the longest time trying to figure out how to accept chains of .pixel_red and pixel references. Once we took a step back and decided .pixel_red should be the parent, with the thing we are calling .pixel_red on as the child, we were able to fix up our grammar and AST right away.

7.1.3 Stick to one syntax for intrinsics

It would have been better to just have one syntax for intrinsics. We used both "load(…)" syntax as well as "p.pixel_red". If we had just chosen the function-like syntax and any arbitrary function argument list for our language intrinsics, we could have had one generic intrinsic function statement and maintained a list of reserved intrinsic function names. Our lexer would not have been able to enforce the number of arguments for each intrinsic function call, but that seems like a reasonable tradeoff for simple extensibility.

7.1.4 Two person teams worked well

We found working in a two person team ideal. When we got stuck, we had another person to shoot ideas off of and to help troubleshoot. It was also nice that we could each concentrate on our individual pieces without worry about other components of the system. Despite this, it was easy getting up to speed with the other's work when it came time for integration, which happened often.

7.1.5 Separate walker and interpreter for easier testing

The walker implementation started out as a walker that simply printed out something similar to the program text. When integration happened, the print statements were completely ripped out and replaced with code that directly called the Java Glimmer libraries. It would have been nicer if we had defined a clear interpreter interface. Then we could have mocked out this component

which would have allowed us to test the walker code itself in isolation. As is, we only have integration testing on the walker.

7.1.6 Writing a Compiler isn't as bad as its reputation

PLT wasn't as bad as we had heard. It was certainly challenging and a decent bit of work, but still did not live up to all the horror stories. We're glad we did it—and we're glad we are finished!

7.1.7 If there had been more time…

These are some additional things we would have liked to have done if there had been more time:

- More testing—is there ever enough?
- We were pretty close to allowing a real-time interpreter environment (such as Python's). We would have had to find a way to order the parser to read in lines at a time and build the AST as it went along, but it seems fairly reasonable to do.
- More fun features—it would have been fun to have more visual image intrinsics, such as for displaying images. There are also a number of language features that would have been useful (iterating over rows only, etc.). But that's really a tangent to what the class is about.

8 Appendix

8.1 Antlr Lexer and Parser Code – grammar.g

```
header {
import java.io.FileInputStream;

import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
}

//--------------------------------------------------------------------------
// GlimmerLexer
// Author: Terry Tai
//--------------------------------------------------------------------------
class GlimmerLexer extends Lexer;
options {
  charVocabulary = '\3'..'\377';
  testLiterals = false;    // Don't automatically test for keywords,
literals, etc.
  k = 2;                   // Lookahead buffer.
}

{
  // Define a main function for testing the lexer.
  public static void main(String[] args) {
    GlimmerLexer myLexer = new GlimmerLexer(System.in);
    while(true) {
      try {
        Token t = myLexer.nextToken();
        System.out.println("Token: " + t.toString());
      } catch(Exception e) {
        System.out.println("Exception: " + e.toString());
        return;
      }
    }
  }
}

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

ID options { testLiterals = true; }
  : LETTER (LETTER | DIGIT | UNDERSCORE)*
;

NUM
  : (DIGIT)+ ('.' (DIGIT)*)?
;

STR
  : '"'!
```

```
    ( ('\\' '"') => '\\'! '"' |
      ~('"')
    )*
    '"'!
;

// Whitespace -- ignored
WS
  : ( ' '
    | '\t'
    | '\f'
    | ( "\r\n"  // DOS/Windows
      | '\r'    // Macintosh
      | '\n'    // Unix
      )
      { newline(); }  // These three are the newline characters.
    )
    { $setType(Token.SKIP); }
;

// Operators
DOT :      '.'  ;
SEMI :     ';'  ;
COMMA :    ','  ;
LBRKT :    '['  ;
RBRKT :    ']'  ;
LPARN :    '('  ;
RPARN :    ')'  ;
LBRACE:    '{'  ;
RBRACE:    '}'  ;
EQ :       "==" ;
NE :       "!=" ;
LT :       '<'  ;
LE :       "<=" ;
GT :       '>'  ;
GE :       ">=" ;
AND :      "&&" ;
OR :       "||" ;
NOT :      '!'  ;
PLUS :     '+'  ;
MINUS :    '-'  ;
TIMES :    '*'  ;
DIV :      '/'  ;
MOD :      '%'  ;
SET :      '='  ;
ALIAS:     "~"  ;

protected DIGIT
  : '0'..'9'
;

protected LETTER
  : ('A'..'Z' | 'a'..'z')
;

protected UNDERSCORE
  : '_'
```

```
;

//-------------------------------------------------------------------------
----
// Glimmer Parser
// Authors: Vida Ha   (Grammar Acceptance for the Parser)
//          Terry Tai (AST Generation lines)
//-------------------------------------------------------------------------
----
class GlimmerParser extends Parser;
options {
  // defaultErrorHandler = true;  // What does this do?
  buildAST = true;
  k = 2;
}

tokens {
  PROGRAM;
  INCLUDE_LIST;
  FUNC_LIST;
  STMT_LIST;
  FUNC_ARGS;
  FUNC_CALL;
  PIXEL_CREATE;
  UPLUS;
  UMINUS;
}

{
  private boolean parserError = false;

  /**
   * Argument 1: input program source filename
   * Argument 2: ast to enable graphical AST display
   */
  public static void main(String[] args) {
    String filename = args[0];
    boolean showast = false;
    if (args.length > 1) {
      if (args[1].equals("ast"))
        showast = true;
    }

    try {
      FileInputStream input = new FileInputStream(filename);
      GlimmerLexer lexer = new GlimmerLexer(input);
      GlimmerParser parser = new GlimmerParser(lexer);
      parser.program();

      CommonAST parseTree = (CommonAST)parser.getAST();

      // Print the AST in a human-readable format.
      System.out.println(parseTree.toStringList());

      // Open a window in which the AST is displayed graphically.
      if (showast) {
```

```
        ASTFrame frame = new ASTFrame("AST from the Glimmer Parser",
parseTree);
        frame.setVisible(true);
      }
    }
    catch (Exception e) {
      System.err.println("parser exception: " + e);
      e.printStackTrace();
    }
  }

  // @Overrides
  public void reportError(RecognitionException e) {
    super.reportError(e);
    parserError = true;
  }

  public boolean hasParserError() {
    return parserError;
  }
}

// MAIN PROGRAM DEFINITION
program
  : include_list func_list stmt_list EOF!
    { #program = #([PROGRAM, "PROGRAM"], program); }
;

// INCLUDES
include_list
  : (include_stmt)*
    { #include_list = #([INCLUDE_LIST, "INCLUDE_LIST"], include_list); }
;

include_stmt
  : "include"^ STR
;

// FUNCTIONS
func_list
  : (func_def)*
    { #func_list = #([FUNC_LIST, "FUNC_LIST"], func_list); }
;

func_def
  : "function"^ ID LPARN! func_args RPARN!
     LBRACE! stmt_list RBRACE!
;

func_args
  : (expression (COMMA! expression)*)?
    { #func_args = #([FUNC_ARGS, "FUNC_ARGS"], func_args); }
;

// STATEMENTS
stmt_list
  : (stmt)*
```

```
    { #stmt_list = #([STMT_LIST, "STMT_LIST"], stmt_list); }
;

stmt
  : set_stmt
  | alias_stmt
  | return_stmt
  | scan_stmt
  | if_else_stmt
  | func_call_stmt
  | load_stmt
  | save_stmt
  | new_stmt
  | print_stmt
  | print_str_stmt
  | break_stmt
  | continue_stmt
;

set_stmt
  : lvalue SET^ expression SEMI!
;

alias_stmt
  : ID ALIAS^ expression SEMI!
;

return_stmt
  : "return"^ (expression)? SEMI!
;

scan_stmt
  : "scan"^ LPARN! ID "in"! expression RPARN!
     LBRACE! stmt_list RBRACE!
;

if_else_stmt
  : "if"^ LPARN! expression RPARN!
     LBRACE! stmt_list RBRACE!
     ("else"! LBRACE! stmt_list RBRACE!)?
;

func_call_stmt
  : func_call SEMI!
;

func_call
  : ID LPARN! func_args RPARN!
  { #func_call = #([FUNC_CALL, "FUNC_CALL"], func_call); }
;

load_stmt
  : "load"^ LPARN! ID COMMA! STR RPARN! SEMI!
;

print_stmt
  : "print"^ LPARN! expression RPARN! SEMI!
```

```
;

print_str_stmt
  : "printstr"^ LPARN! STR RPARN! SEMI!
;

break_stmt
  : "break"^ SEMI!
;

continue_stmt
  : "continue"^ SEMI!
;

save_stmt
  : "save"^ LPARN! ID COMMA! STR RPARN! SEMI!
;

new_stmt
  : "new"^ LPARN! ID COMMA! expression COMMA! expression RPARN! SEMI!
;

// PIXEL/IMAGE STUFF
lvalue
  : pixel_ref
     (DOT! ("pixel_red"^ | "pixel_green"^ | "pixel_blue"^ | "pixel_alpha"^
            | "pixel_x"^ | "pixel_y"^ | "image_height"^ | "image_width"^) )?
;

pixel_ref
  : ID (LBRKT^ expression COMMA! expression RBRKT!)*
;

pixel_create
  : LBRKT! NUM COMMA! NUM COMMA! NUM COMMA! NUM RBRKT!
  { #pixel_create = #([PIXEL_CREATE, "PIXEL_CREATE"], pixel_create); }
;

// EXPRESSIONS
expression
  : logic_term ( OR^ logic_term )*
;

logic_term
  : logic_factor ( AND^ logic_factor )*
;

logic_factor
  : (NOT^)? comparison_expr
;

comparison_expr
  : arith_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NE^) arith_expr )?
;

arith_expr
  : arith_term ( (PLUS^ | MINUS^) arith_term )*
```

```
;

arith_term
  : arith_factor ( (TIMES^ | DIV^ | MOD^) arith_factor )*
;

arith_factor
  : PLUS! atom
      {#arith_factor = #([UPLUS,"UPLUS"], arith_factor); }
  | MINUS! atom
      {#arith_factor = #([UMINUS,"UMINUS"], arith_factor); }
  | atom
;

atom
  : LPARN! expression RPARN!
  | NUM
  | pixel_create
  | lvalue
  | func_call
;
```

8.2 Java Library code

8.2.1 GlimmerSymbolTable.java

```
import java.util.Hashtable;
import java.util.Enumeration;

/**
 * Glimmer's symbol table will contain essentially a pointer to the parent as
 * well as a Hashtable to hold the values at this level.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerSymbolTable {

  private Hashtable table_;
  private GlimmerSymbolTable parent_;

  public GlimmerSymbolTable() {
    table_ = new Hashtable();
    parent_ = null;
  }

  @Override
  public String toString() {
    String ret = "Symbol Table:\n";

    Enumeration keys = table_.keys();
    while (keys.hasMoreElements()) {
      String key = (String)keys.nextElement();
      GlimmerType obj = (GlimmerType)(get(key));
      ret += (key + ": " + obj + "\n");
```

```
    }

    return ret;
  }

  private void setParent(GlimmerSymbolTable parent) {
    parent_ = parent;
  }

  // Creates a child scope, which the current table set as the parent.
  public GlimmerSymbolTable openScope() {
    GlimmerSymbolTable child_table = new GlimmerSymbolTable();
    child_table.setParent(this);
    return child_table;
  }

  // Returns a pointer to the parent - this is how we close a scope.
  public GlimmerSymbolTable getParent() {
    return parent_;
  }

  public void add(String key, GlimmerType value) {
    table_.put(key, value);
  }

  // Gets the object bound to the key in this level of the table only.
  public GlimmerType getLocal(String key) {
    return (GlimmerType)(table_.get(key));
  }

  // Recursively moves up the symbol table ladder until we find a valid
  // instance of the key (or until we run out).
  public GlimmerType get(String key) {
    GlimmerType thing = getLocal(key);
    if (thing == null && parent_ != null)
      thing = parent_.get(key);
    return thing;
  }

  public Hashtable getTable() {
    return table_;
  }

  public void mergeFunctions(GlimmerSymbolTable new_table) throws
GlimmerException {
    Hashtable new_ht = new_table.getTable();
    Enumeration keys = new_ht.keys();
    while (keys.hasMoreElements()) {
      String key = (String)keys.nextElement();
      GlimmerFunction new_func = (GlimmerFunction)new_table.get(key);
      GlimmerFunction old_func = (GlimmerFunction)get(key);
      if (old_func != null) {
        if (!old_func.equals(new_func))
          throw new GlimmerException("Function " + key + " was defined
differently in alternate file.");
      } else {
        add(key, new_func);
```

```
      }
    }
  }

}
```

8.2.2 GlimmerException.java

```
/**
 * Basic exception handling for Glimmer.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerException extends java.lang.Exception {
  public GlimmerException(String msg) {
    super(msg);
  }
}
```

8.2.3 GlimmerType.java

```
import java.io.*;
import java.awt.image.*;

enum ControlType {
  RUN,
  RUN_LOOP,
  RUN_FUNC,
  RETURN,
  BREAK,
  CONTINUE,
  DO_NOT_RUN,
}

/**
 * GlimmerType defines the various primal types supported by Glimmer.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerType {
  public GlimmerType() {
  }

  public String toString() {
    return "<uninitialized object>";
  }

  public String typeName() {
    return "BasicType";
  }

  public boolean validRValue() {
    return false;
  }
```

```
  // Operations that can be performed on variables.  Subclasses should define
  // their own versions for whatever is appropriate.
  public GlimmerType copy() throws GlimmerException {
    throw new GlimmerException("Copy unsupported for " + toString() + ".");
  }
  public void replace(GlimmerType base) throws GlimmerException {
  }

  public GlimmerType add(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Addition unsupported for " + toString() +
"."");
  }
  public GlimmerType sub(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Subtraction unsupported for " + toString() +
"."");
  }
  public GlimmerType mul(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Multiplication unsupported for " + toString()
                               + ".");
  }
  public GlimmerType div(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Division unsupported for " + toString() +
"."");
  }
  public GlimmerType mod(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Modulus unsupported for " + toString() +
"."");
  }

  public GlimmerType neg() throws GlimmerException {
    throw new GlimmerException("Unary-Negative unsupported for " + toString()
                               + ".");
  }
  public GlimmerType not() throws GlimmerException {
    throw new GlimmerException("Logical-Negation unsupported for " +
toString()
                               + ".");
  }

  public GlimmerType lt(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Less-Than unsupported for " + toString() +
"."");
  }
  public GlimmerType gt(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Greater-Than unsupported for " + toString() +
"."");
  }
  public GlimmerType le(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Less-Than-Or-Equal unsupported for " +
toString()
                               + ".");
  }
  public GlimmerType ge(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Greater-Than-Or-Equal unsupported for " +
                               toString() + ".");
  }
```

```
  public GlimmerType eq(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Equals unsupported for " + toString() + ".");
  }
  public GlimmerType ne(GlimmerType right) throws GlimmerException {
    throw new GlimmerException("Not-Equals unsupported for " + toString() +
".");
  }

  public GlimmerType pixelRef(int x, int y) throws GlimmerException {
    throw new GlimmerException("Pixel-Reference unsupported for " +
toString()
                               + ".");
  }
  public GlimmerType pixelRed() throws GlimmerException {
    throw new GlimmerException("Pixel-Red unsupported for " + toString() +
".");
  }
  public GlimmerType pixelGreen() throws GlimmerException {
    throw new GlimmerException("Pixel-Green unsupported for " + toString() +
".");
  }
  public GlimmerType pixelBlue() throws GlimmerException {
    throw new GlimmerException("Pixel-Blue unsupported for " + toString() +
".");
  }
  public GlimmerType pixelAlpha() throws GlimmerException {
    throw new GlimmerException("Pixel-Alpha unsupported for " + toString() +
".");
  }
  public GlimmerType pixelX() throws GlimmerException {
    throw new GlimmerException("Pixel-X-Position unsupported for " +
toString()
                               + ".");
  }
  public GlimmerType pixelY() throws GlimmerException {
    throw new GlimmerException("Pixel-Y-Position unsupported for " +
toString()
                               + ".");
  }

  public GlimmerType imageHeight() throws GlimmerException {
    throw new GlimmerException("Image-Height unsupported for " + toString() +
".");
  }
  public GlimmerType imageWidth() throws GlimmerException {
    throw new GlimmerException("Image-Width unsupported for " + toString() +
".");
  }
}
```

8.2.4 GlimmerID.java

```
/**
 * GlimmerID is a variable name storehouse -- of size 1.
 *
```

```
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerID extends GlimmerType {
  private String id_name_;

  public GlimmerID() {
  }

  @Override
  public String typeName() {
    return "<unbound identifier: " + id_name_ + ">";
  }
  @Override
  public String toString() {
    return id_name_;
  }

  public void setID(String id) {
    id_name_ = id;
  }
  public String getID() {
    return id_name_;
  }
}
```

8.2.5 GlimmerNum.java

```
enum NumberField {
  NONE,
  PIXEL_RED,
  PIXEL_GREEN,
  PIXEL_BLUE,
  PIXEL_ALPHA,
  PIXEL_X,
  PIXEL_Y,
  IMAGE_HEIGHT,
  IMAGE_WIDTH,
}

/**
 * GlimmerNum is the basic number type for the Glimmer language.  It is
 * internally represented as a Java-style float with all the associated
issues
 * (and benefits?).
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerNum extends GlimmerType {
  private float value_;         // The value of this number.

  private GlimmerPixel pixel_;  // Only one of these should be non-null -
this is
  private GlimmerImage image_;  // who this number is associated with.
  private NumberField field_;   // Which field this number is for.
```

```
public final static GlimmerNum TRUE_VALUE = new GlimmerNum(1.0f);
public final static GlimmerNum FALSE_VALUE = new GlimmerNum(0.0f);

public GlimmerNum(float value) {
  value_ = value;
  pixel_ = null;
  image_ = null;
  field_ = NumberField.NONE;
}

public void setPixel(GlimmerPixel pixel, NumberField field) {
  pixel_ = pixel;
  image_ = null;
  field_ = field;
}

public void setImage(GlimmerImage image, NumberField field) {
  pixel_ = null;
  image_ = image;
  field_ = field;
}

@Override
public String toString() {
  String retstr = null;
  try {
    retstr = String.valueOf(getValue());
  } catch (Exception e) {
    System.err.println(e.getMessage());
    assert(false);
  }
  if (retstr.endsWith(".0"))
    return retstr.substring(0, retstr.length() - 2);
  return retstr;
}
@Override
public String typeName() {
  return "Num";
}
@Override
public boolean validRValue() {
  return true;
}
@Override
public GlimmerType copy() throws GlimmerException {
  GlimmerNum new_num = new GlimmerNum(getValue());
  return new_num;
}

public float getValue() throws GlimmerException {
  if (pixel_ != null) {
    switch (field_) {
      case PIXEL_RED:    return pixel_.pixelRed().getPureValue();
      case PIXEL_GREEN:  return pixel_.pixelGreen().getPureValue();
      case PIXEL_BLUE:   return pixel_.pixelBlue().getPureValue();
      case PIXEL_ALPHA:  return pixel_.pixelAlpha().getPureValue();
      case PIXEL_X:      return pixel_.pixelX().getPureValue();
```

```
        case PIXEL_Y:      return pixel_.pixelY().getPureValue();
        default:           throw new GlimmerException("Invalid pixel field "
+
                                                      field_ +
                                                      " in
GlimmerNum.getValue()");
      }
    }
    if (image_ != null) {
      switch (field_) {
        case IMAGE_WIDTH:   return (float)image_.imageWidth_internal();
        case IMAGE_HEIGHT:  return (float)image_.imageHeight_internal();
        default:            throw new GlimmerException(
                              "Invalid image field " + field_ +
                              " in " + GlimmerNum.getValue()");
      }
    }
    return value_;
  }

  public float getPureValue() {
    return value_;
  }

  public NumberField getField() {
    return field_;
  }

  @Override
  public void replace(GlimmerType base) throws GlimmerException {
    GlimmerNum num = (GlimmerNum)base;
    if (pixel_ != null) {
      pixel_.changeComponent(num, getField());
    } else if (image_ != null) {
      image_.resizeDimension(num, getField());
    } else {
      value_ = num.getValue();
    }
  }

  public void replacePure(GlimmerType base) throws GlimmerException {
    value_ = ((GlimmerNum)base).getValue();
  }

  @Override
  public GlimmerType add(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot add non-number to number.");
    return new GlimmerNum(getValue() + ((GlimmerNum)right).getValue());
  }
  @Override
  public GlimmerType sub(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot subtract non-number from number");
    return new GlimmerNum(getValue() - ((GlimmerNum)right).getValue());
  }
  @Override
```

```
  public GlimmerType mul(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot multiply non-number with number.");
    return new GlimmerNum(getValue() * ((GlimmerNum)right).getValue());
  }
  @Override
  public GlimmerType div(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot divide number by non-number.");
    return new GlimmerNum(getValue() / ((GlimmerNum)right).getValue());
  }
  @Override
  public GlimmerType mod(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot take modulus of number by non-
number.");
    return new GlimmerNum(getValue() % ((GlimmerNum)right).getValue());
  }

  @Override
  public GlimmerType neg() throws GlimmerException {
    return new GlimmerNum(-getValue());
  }
  @Override
  public GlimmerType not() throws GlimmerException {
    if (getValue() == 0.0f)
      return new GlimmerNum(1.0f);
    return new GlimmerNum(0.0f);
  }
  public boolean isTrue() throws GlimmerException {
    return getValue() != 0.0f;
  }

  @Override
  public GlimmerType lt(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot compare number with non-number.");
    if (getValue() < ((GlimmerNum)right).getValue())
      return new GlimmerNum(1.0f);
    return new GlimmerNum(0.0f);
  }
  @Override
  public GlimmerType gt(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot compare number with non-number.");
    if (getValue() > ((GlimmerNum)right).getValue())
      return new GlimmerNum(1.0f);
    return new GlimmerNum(0.0f);
  }
  @Override
  public GlimmerType le(GlimmerType right) throws GlimmerException {
    return gt(right).not();
  }
  @Override
  public GlimmerType ge(GlimmerType right) throws GlimmerException {
    return lt(right).not();
  }
```

```
  @Override
  public GlimmerType eq(GlimmerType right) throws GlimmerException {
    if (!(right instanceof GlimmerNum))
      throw new GlimmerException("Cannot compare number with non-number.");
    if (getValue() == ((GlimmerNum)right).getValue())
      return new GlimmerNum(1.0f);
    return new GlimmerNum(0.0f);
  }
  @Override
  public GlimmerType ne(GlimmerType right) throws GlimmerException {
    return eq(right).not();
  }

}
```

8.2.6 GlimmerPixel.java

```
/**
 * GlimmerPixel is the representation of a pixel, either associated or not
with
 * a GlimmerImage.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerPixel extends GlimmerType {
  GlimmerNum r_, g_, b_, a_;
  int x_, y_;
  GlimmerImage image_;

  public GlimmerPixel(int r, int g, int b, int alpha) {
    r_ = new GlimmerNum(r);
    r_.setPixel(this, NumberField.PIXEL_RED);
    g_ = new GlimmerNum(g);
    g_.setPixel(this, NumberField.PIXEL_GREEN);
    b_ = new GlimmerNum(b);
    b_.setPixel(this, NumberField.PIXEL_BLUE);
    a_ = new GlimmerNum(alpha);
    a_.setPixel(this, NumberField.PIXEL_ALPHA);
    setImageAttr(null, -1, -1);
  }

  public GlimmerPixel(GlimmerImage image, int x, int y) {
    r_ = g_ = b_ = a_ = null;
    setImageAttr(image, x, y);
  }

  @Override
  public String typeName() {
    return "Pixel";
  }
  @Override
  public String toString() {
    String retstr = (image_ != null) ? "Image(" + x_ + "," + y_ + "):" : "";
    try {
```

```
      retstr += "(" + pixelRed() + "," + pixelGreen() + "," + pixelBlue() +
")/" +
                pixelAlpha();
    } catch (GlimmerException ge) {
      retstr += "<GlimmerException while obtaining colors: " +
                ge.getMessage() + ">";
    }
    return retstr;
  }
  @Override
  public boolean validRValue() {
    return true;
  }
  @Override
  public GlimmerType copy() throws GlimmerException {
    GlimmerPixel pix =
      new GlimmerPixel((int)((GlimmerNum)pixelRed()).getValue(),
                       (int)((GlimmerNum)pixelGreen()).getValue(),
                       (int)((GlimmerNum)pixelBlue()).getValue(),
                       (int)((GlimmerNum)pixelAlpha()).getValue());
    return pix;
  }

  public void setImageAttr(GlimmerImage image, int x, int y) {
    image_ = image;
    x_ = x;
    y_ = y;
  }
  public GlimmerImage getImage() {
    return image_;
  }

  @Override
  public GlimmerType eq(GlimmerType right) throws GlimmerException {
    // This will return true (1.0) if the colors and transparencies are the
    // same AND if the image components are the same (but only if they're
    // defined in both).
    if (!(right instanceof GlimmerPixel))
      throw new GlimmerException("Cannot compare pixel with non-pixel.");
    GlimmerPixel rpix = (GlimmerPixel)right;
    if (pixelRed().getValue() == rpix.pixelRed().getValue() &&
        pixelGreen().getValue() == rpix.pixelGreen().getValue() &&
        pixelBlue().getValue() == rpix.pixelBlue().getValue() &&
        pixelAlpha().getValue() == rpix.pixelAlpha().getValue())
      return new GlimmerNum(1.0f);
    return new GlimmerNum(0.0f);
  }
  @Override
  public GlimmerType ne(GlimmerType right) throws GlimmerException {
    GlimmerNum eq_num = (GlimmerNum)eq(right);
    return eq_num.not();
  }

  @Override
  public GlimmerType pixelRef(int x, int y) throws GlimmerException {
    return image_.pixelRef(x_ + x, y_ + y);
  }
```

```
  @Override
  public GlimmerNum pixelRed() throws GlimmerException {
    if (image_ == null)
      return r_;
    GlimmerNum num = new GlimmerNum(pixelImageRed());
    num.setPixel(this, NumberField.PIXEL_RED);
    return num;
  }
  @Override
  public GlimmerNum pixelGreen() throws GlimmerException {
    if (image_ == null)
      return g_;
    GlimmerNum num = new GlimmerNum(pixelImageGreen());
    num.setPixel(this, NumberField.PIXEL_GREEN);
    return num;
  }
  @Override
  public GlimmerNum pixelBlue() throws GlimmerException {
    if (image_ == null)
      return b_;
    GlimmerNum num = new GlimmerNum(pixelImageBlue());
    num.setPixel(this, NumberField.PIXEL_BLUE);
    return num;
  }
  @Override
  public GlimmerNum pixelAlpha() throws GlimmerException {
    if (image_ == null)
      return a_;
    GlimmerNum num = new GlimmerNum(pixelImageAlpha());
    num.setPixel(this, NumberField.PIXEL_ALPHA);
    return num;
  }
  @Override
  public GlimmerNum pixelX() throws GlimmerException {
    if (image_ == null)
      throw new GlimmerException("Cannot take pixel_x of an unbound pixel.");
    GlimmerNum num = new GlimmerNum(x_);
    num.setPixel(this, NumberField.PIXEL_X);
    return num;
  }
  @Override
  public GlimmerNum pixelY() throws GlimmerException {
    if (image_ == null)
      throw new GlimmerException("Cannot take pixel_y of an unbound pixel.");
    GlimmerNum num = new GlimmerNum(y_);
    num.setPixel(this, NumberField.PIXEL_Y);
    return num;
  }

  public int pixelImageRed() throws GlimmerException {
    if (image_ == null)
      throw new GlimmerException("Cannot get red img info from unbound
pixel.");
    int[] colors = image_.getColors(x_, y_);
    return colors[0];
  }
  public int pixelImageGreen() throws GlimmerException {
```

```
    if (image_ == null)
      throw new GlimmerException("Cannot get green img info from unbound
pixel.");
    int[] colors = image_.getColors(x_, y_);
    return colors[1];
  }
  public int pixelImageBlue() throws GlimmerException {
    if (image_ == null)
      throw new GlimmerException("Cannot get blue img info from unbound
pixel.");
    int[] colors = image_.getColors(x_, y_);
    return colors[2];
  }
  public int pixelImageAlpha() throws GlimmerException {
    if (image_ == null)
      throw new GlimmerException("Cannot get alpha img info from unbound
pixel.");
    int[] colors = image_.getColors(x_, y_);
    return colors[3];
  }
  public int pixelImageX() throws GlimmerException {
    return x_;
  }
  public int pixelImageY() throws GlimmerException {
    return y_;
  }

  public GlimmerType getNext() throws GlimmerException {
    if (image_ == null)
      throw new GlimmerException("Cannot iterate from image-unbound pixel.");
    else
      return image_.getNextPixel(x_, y_);
  }

  public void changeComponent(GlimmerNum num, NumberField field)
      throws GlimmerException {
    GlimmerPixel newcolors;
    switch (field) {
      case PIXEL_RED: {
        newcolors = new GlimmerPixel((int)num.getValue(),
                                     (int)pixelGreen().getValue(),
                                     (int)pixelBlue().getValue(),
                                     (int)pixelAlpha().getValue());
        break;
      }
      case PIXEL_GREEN: {
        newcolors = new GlimmerPixel((int)pixelRed().getValue(),
                                     (int)num.getValue(),
                                     (int)pixelBlue().getValue(),
                                     (int)pixelAlpha().getValue());
        break;
      }
      case PIXEL_BLUE: {
        newcolors = new GlimmerPixel((int)pixelRed().getValue(),
                                     (int)pixelGreen().getValue(),
                                     (int)num.getValue(),
                                     (int)pixelAlpha().getValue());
```

```
        break;
      }
      case PIXEL_ALPHA: {
        newcolors = new GlimmerPixel((int)pixelRed().getValue(),
                                     (int)pixelGreen().getValue(),
                                     (int)pixelBlue().getValue(),
                                     (int)num.getValue());
        break;
      }
      default: {
        throw new GlimmerException("Cannot change pixel attribute: " +
                                   num.getField());
      }
    }

    replace(newcolors);
  }

  @Override
  public void replace(GlimmerType base) throws GlimmerException {
    GlimmerPixel pixel = (GlimmerPixel)base;
    if (image_ == null) {
      r_.replacePure(pixel.pixelRed());
      g_.replacePure(pixel.pixelGreen());
      b_.replacePure(pixel.pixelBlue());
      a_.replacePure(pixel.pixelAlpha());
    } else {
      image_.changePixel(pixel, pixelImageX(), pixelImageY());
    }
  }
}
```

8.2.7 GlimmerImage.java

```
import java.io.*;
import java.awt.image.*;
import org.cmc.sanselan.*;

/**
 * GlimmerImage is the image primal type.  This is based on the Sanselan
image
 * library for writing and reading, and otherwise on Java's native stuff.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerImage extends GlimmerType {
  BufferedImage pic_;
  GlimmerPixel pixels_[][];

  public GlimmerImage() {
  }

//  public GlimmerImage(File file, BufferedImage pic) {
//    file_ = file;
//    pic_ = pic;
```

```
//  }

  public GlimmerImage(String filename) throws GlimmerException {
    intrinsicLoad(filename);
  }

  // Returns an array of ints corresponding to the red, green, blue, and
alpha
  // values at that location.
  public int[] getColors(int x, int y) {
    int argb = pic_.getRGB(x, y);
    int[] array = new int[4];
    array[0] = (int)((argb >> 16) & 0xFF);
    array[1] = (int)((argb >> 8) & 0xFF);
    array[2] = (int)(argb & 0xFF);
    array[3] = (int)((argb >> 24) & 0xFF);
    return array;
  }

  @Override
  public String typeName() {
    return "Image";
  }
  @Override
  public String toString() {
    try {
      return "Image: " + imageHeight() + " rows by " + imageWidth() + "
columns";
    } catch (Exception e) {
      System.err.println(e.getMessage());
      assert(false);
    }
    return null;
  }
  @Override
  public boolean validRValue() {
    return true;
  }
  @Override
  public GlimmerType copy() throws GlimmerException {
    GlimmerImage new_img = new GlimmerImage();
    new_img.replace(this);
    return new_img;
  }

  // Returns the next pixel in the image from the current position.  Returns
  // null if we're "done" (i.e. outside the bounds).  Uses row-major
ordering.
  public GlimmerType getNextPixel(int curr_x, int curr_y) throws
GlimmerException {
    try {
      if (++curr_x >= pic_.getWidth()) {
        curr_x = 0;
        ++curr_y;
      }
      if (curr_y >= pic_.getHeight())
        return null;
```

```
      return pixelRef(curr_x, curr_y);
    } catch (Exception e) {
      throw new GlimmerException(e.getMessage());
    }
  }

  @Override
  public GlimmerType pixelRef(int x, int y) throws GlimmerException {
    return pixels_[y][x];
  }

  @Override
  public GlimmerType imageHeight() throws GlimmerException {
    GlimmerNum num = new GlimmerNum(imageHeight_internal());
    num.setImage(this, NumberField.IMAGE_HEIGHT);
    return num;
  }
  @Override
  public GlimmerType imageWidth() throws GlimmerException {
    GlimmerNum num = new GlimmerNum(imageWidth_internal());
    num.setImage(this, NumberField.IMAGE_WIDTH);
    return num;
  }

  public int imageHeight_internal() throws GlimmerException {
    return pic_.getHeight();
  }
  public int imageWidth_internal() throws GlimmerException {
    return pic_.getWidth();
  }

  public void resizeDimension(GlimmerNum num, NumberField field)
      throws GlimmerException {
    if (field == NumberField.IMAGE_HEIGHT)
      resize(imageWidth_internal(), (int)num.getValue());
    else if (field == NumberField.IMAGE_WIDTH)
      resize((int)num.getValue(), imageHeight_internal());
    else
      throw new GlimmerException("Cannot resize image using " +
num.toString());
  }

  public GlimmerType intrinsicNew(int size_x, int size_y) throws
GlimmerException {
    try {
      pic_ = new BufferedImage(size_x, size_y, BufferedImage.TYPE_INT_ARGB);
      InitPixels();
    } catch (Exception e) {
      throw new GlimmerException(e.getMessage());
    }
    return this;
  }

  public void intrinsicLoad(String filename) throws GlimmerException {
    try {
      File file = new File(filename);
      pic_ = Sanselan.getBufferedImage(file);
```

```
      InitPixels();
    } catch (Exception e) {
      throw new GlimmerException(e.getMessage());
    }
  }

  public void intrinsicWrite(String filename) throws GlimmerException {
    try {
      File outfile = new File(filename);
      // null = options map.
      Sanselan.writeImage(pic_, outfile, ImageFormat.IMAGE_FORMAT_PNG, null);
    } catch (Exception e) {
      throw new GlimmerException(e.getMessage());
    }
  }

  public void changePixel(GlimmerPixel pixel, int x, int y) throws
GlimmerException {
    int value = (((int)pixel.pixelAlpha().getValue()) << 24) +
                ((int)(pixel.pixelRed().getValue()) << 16) +
                ((int)(pixel.pixelGreen().getValue()) << 8) +
                ((int)pixel.pixelBlue().getValue());
    pic_.setRGB(x, y, value);
  }

  private void InitPixels() throws GlimmerException {
    pixels_ = new
GlimmerPixel[imageHeight_internal()][imageWidth_internal()];
    for (int y = 0; y < imageHeight_internal(); ++y) {
      for (int x = 0; x < imageWidth_internal(); ++x) {
        pixels_[y][x] = new GlimmerPixel(this, x, y);
      }
    }
  }

  @Override
  public void replace(GlimmerType base) throws GlimmerException {
    if (!(base instanceof GlimmerImage))
      throw new GlimmerException("Images can only be replaced by images.");
    GlimmerImage img = (GlimmerImage)base;

    try {
      pic_ = new
BufferedImage((int)((GlimmerNum)(img.imageWidth())).getValue(),

(int)((GlimmerNum)(img.imageHeight())).getValue(),
                                BufferedImage.TYPE_INT_ARGB);
      pixels_ = new
GlimmerPixel[imageHeight_internal()][imageWidth_internal()];
      for (int y = 0; y < imageHeight_internal(); ++y) {
        for (int x = 0; x < imageWidth_internal(); ++x) {
          pic_.setRGB(x, y, img.pic_.getRGB(x, y));
          pixels_[y][x] = new GlimmerPixel(this, x, y);
        }
      }
    } catch (Exception e) {
      throw new GlimmerException(e.getMessage());
```

```
    }
  }

  public void resize(int new_x, int new_y) throws GlimmerException {
    try {
      BufferedImage new_pic = new BufferedImage(new_x, new_y,
                                                BufferedImage.TYPE_INT_ARGB);
      GlimmerPixel new_pixels[][] = new GlimmerPixel[new_y][new_x];
      for (int y = 0; y < new_y; ++y) {
        for (int x = 0; x < new_x; ++x) {
          if (y < imageHeight_internal() && x < imageWidth_internal())
            new_pic.setRGB(x, y, pic_.getRGB(x, y));
          new_pixels[y][x] = new GlimmerPixel(this, x, y);
        }
      }
      pic_ = new_pic;
      pixels_ = new_pixels;
    } catch (Exception e) {
      throw new GlimmerException(e.getMessage());
    }
  }

}
```

8.2.8 GlimmerFunction.java

```
import antlr.collections.AST;
import java.util.Arrays;

/**
 * GlimmerFunction stores the Antlr AST so we can run functions later.
 *
 * @author netherlight@gmail.com (Terry Tai)
 */
public class GlimmerFunction extends GlimmerType {
  private AST ast_;              // The Antlr abstract symbol tree.
  private String[] parameters_;  // The local variable names for the
arguments.

  public GlimmerFunction(String[] params, AST ast) {
    ast_ = ast;
    parameters_ = params;
  }

  @Override
  public String toString() {
    String str = "GlimmerFunction:\n";
    for (String param : parameters_) {
      str += param;
      str += " ";
    }
    str += ("->\n" + ast_);
    return str;
  }
```

```
  @Override
  public String typeName() {
    return "Function";
  }

  public AST getAST() {
    return ast_;
  }

  public String[] getParams() {
    return parameters_;
  }

  @Override
  public void replace(GlimmerType base) throws GlimmerException {
    ast_ = ((GlimmerFunction)base).getAST();
  }

  public boolean equals(GlimmerFunction func) {
    if (!Arrays.equals(parameters_, func.getParams()))
      return false;
    if (!ast_.equalsTree(func.getAST()))
      return false;
    return true;
  }
}
```

8.3 Walker Code – walker.g

```
header {
import java.io.*;
import java.lang.*;
import java.util.*;

import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
}

//----------------------------------------------------------------------------
----
// Glimmer Walker
// Author: Vida Ha
//----------------------------------------------------------------------------
----
class GlimmerWalker extends TreeParser;
options{
    importVocab = GlimmerLexer;
}
{
  /**
   * Argument 1: input program source filename.
   * Argument 2: "ast" to display the AST graphically.
   */
  public static void main(String[] args) {
```

```
    String filename = args[0];

    try {
      FileInputStream input = new FileInputStream(filename);
      GlimmerLexer lexer = new GlimmerLexer(input);
      GlimmerParser parser = new GlimmerParser(lexer);
      parser.program();

      if (parser.hasParserError()) {
        throw new RecognitionException("Will not run with parse errors.");
      }

      CommonAST parseTree = (CommonAST)parser.getAST();

      // Open a window in which the AST is displayed graphically.
      if (args.length > 1 && args[1].equalsIgnoreCase("ast")) {
        ASTFrame frame = new ASTFrame("AST from the Glimmer Parser",
parseTree);
        frame.setVisible(true);
      }

      GlimmerWalker walker = new GlimmerWalker();
      walker.setControlType(ControlType.RUN);
      GlimmerType ret = walker.expr(parseTree);
    }
    catch (Exception e) {
      System.err.println(e);
      assert(false);
    }
  }

  private GlimmerSymbolTable global_table = new GlimmerSymbolTable();
  private GlimmerSymbolTable current_table = global_table;
  private Stack control_stack = new Stack();
  private GlimmerSymbolTable function_table = new GlimmerSymbolTable();
  private static HashSet included_files = new HashSet();

  public void setControlType(ControlType type) {
    control_stack.push(type);
  }

  public GlimmerSymbolTable getFunctionTable() {
    return function_table;
  }
}

expr returns [GlimmerType ret] throws GlimmerException
{
  GlimmerType first = null, second = null, third = null, fourth = null;
  Vector<GlimmerType> args;
  ret = null;
}
  // MAIN PROGRAM DEFINITION
    : #(PROGRAM first=expr second=expr third=expr)
      { ret = third; }

  // INCLDUES
```

```
    | #(INCLUDE_LIST (incl:. { expr(#incl); })* )
    | #("include" file:STR
      {
        try {
        if (included_files.contains(file.toString())) {
          System.out.println("Skipping repeat inclusion of " +
                             file.toString());
        } else {
          included_files.add(file.toString());

          FileInputStream input = new FileInputStream(file.toString());
          GlimmerLexer lexer = new GlimmerLexer(input);
          GlimmerParser parser = new GlimmerParser(lexer);
          parser.program();
          CommonAST parse_tree = (CommonAST)parser.getAST();
          GlimmerWalker walker = new GlimmerWalker();
          walker.setControlType(ControlType.DO_NOT_RUN);
          walker.expr(parse_tree);

          GlimmerSymbolTable file_func_table = walker.getFunctionTable();
          function_table.mergeFunctions(file_func_table);
        }
      } catch (Exception e) {
        throw new GlimmerException(e.getMessage());
        }
      })

  // FUNCTIONS
    | #(FUNC_LIST (func_def:. { expr(#func_def); })* )
    | #("function" fname:ID args=arg_list func_stmts:.
      {
        // Check all the args to make sure they are ID's.
        String[] func_arg_names = new String[args.size()];
        for (int i = 0; i < args.size(); i++) {
          GlimmerType curr = args.elementAt(i);
          if (curr instanceof GlimmerID)
            func_arg_names[i] = ((GlimmerID)curr).getID();
          else
            throw new GlimmerException("Function args must be identifiers: "
+
                                       curr.toString());
        }

        // Create the function.
        GlimmerFunction new_function =
          new GlimmerFunction(func_arg_names, func_stmts);

        // Add the function to the function symbol table.
      GlimmerFunction old_function =
          (GlimmerFunction)(function_table.get(fname.toString()));
        if (old_function != null && !new_function.equals(old_function))
          throw new GlimmerException("Function " + fname.toString() +
                                     " was previously defined differently.");
        function_table.add(fname.toString(), new_function);
      })

  // STATEMENTS
```

```
| #(STMT_LIST (stmt:. {
    ControlType control = (ControlType)control_stack.peek();
    if (control == ControlType.RUN ||
        control == ControlType.RUN_LOOP ||
        control == ControlType.RUN_FUNC)
      ret = expr(#stmt);
  }
  )* )
| #(SET first=expr second=expr)
  {
    // Check that the rvalue is valid.
    if (second == null || !second.validRValue()) {
      throw new GlimmerException(
        "Invalid or undefined variable: " +
        (second != null ? second.toString() : "<null>"));
    }
    // Now set it if appropriate.
    if (first instanceof GlimmerID) {
      // Either create a new instance...
      current_table.add(((GlimmerID)first).getID(), second.copy());
    } else {
      // Otherwise, just mutate the current object.
      first.replace(second);
    }
  }
| #(ALIAS alias:ID second=expr)
  {
    // Check that the rvalue is valid.
    if (second == null || !second.validRValue()) {
      throw new GlimmerException(
        "Invalid or undefined variable: " +
        (second != null ? second.toString() : "<null>"));
    }

    current_table.add(alias.toString(), second);
  }
| #("return" (first=expr)? )
  {
    control_stack.push(ControlType.RETURN);
    if (first != null) {
      if (!first.validRValue())
        throw new GlimmerException("Cannot return invalid expression: " +
                                   first.toString());
      ret = first;
    }
  }
| #("scan" pixel_id:ID second=expr scan_stmts:.
  {
    if ( !(second instanceof GlimmerImage))
       throw new GlimmerException("Cannot scan non-image.");
    else {
      // Iterate through image.
    control_stack.push(ControlType.RUN_LOOP);
      GlimmerImage image = (GlimmerImage)second;
      for (GlimmerPixel pixel = (GlimmerPixel)(image.pixelRef(0,0));
           pixel != null;
           pixel = (GlimmerPixel)(pixel.getNext())) {
```

```
            // Open a new scope.
            current_table = current_table.openScope();
            current_table.add(pixel_id.toString(), pixel);

            // Evaluate stmt list.
            ret = expr(#scan_stmts);

            // Close the old scope.
            current_table = current_table.getParent();

            // At the end of a scan loop, if we wanted to continue, we can
run
            // again.
          ControlType control = (ControlType)control_stack.peek();
            if (control == ControlType.CONTINUE)
            control_stack.pop();
            // If we wanted to break, then we can continue running (so we set
            // this), but then we need to break out of this loop.
            else if (control == ControlType.BREAK) {
            control_stack.pop();
              break;
            }
          else if (control == ControlType.RETURN) {
            break;
          }
          }
        if ((ControlType)control_stack.peek() == ControlType.RUN_LOOP)
          control_stack.pop();
        }
      })
    | #("if" first=expr if_stmts:. (else_stmts:.)?)
      {
        if (!(first instanceof GlimmerNum))
          throw new GlimmerException("If predicate must be a number.");

      // Open a new scope that is a child of the current one.
        current_table = current_table.openScope();

        if (((GlimmerNum)first).isTrue())
          ret = expr(#if_stmts);
        else if (else_stmts != null)
          ret = expr(#else_stmts);

        // Restore old scope.
        current_table = current_table.getParent();
      }
    | #(FUNC_CALL fcall:ID args=arg_list)
      {
        // Get the Glimmer function.
        GlimmerFunction function =
          (GlimmerFunction) (function_table.get(fcall.toString()));
        if (function == null)
          throw new GlimmerException("Cannot call undefined function: " +
                                     fcall.toString());

        // Open up a brand new scope, just for this function.
        GlimmerSymbolTable old_table = current_table;
```

```
        current_table = new GlimmerSymbolTable();

        // Push all the function arguments into the symbol table.
        String[] params = function.getParams();
        if (params.length != args.size()) {
          throw new GlimmerException(
            "Function " + fcall.toString() + " expects " + params.length +
             " argument(s) instead of " + args.size() + ".");
        }
        for (int i = 0; i < params.length; i++) {
          GlimmerType curr = args.elementAt(i);
          // Check to make sure the GlimmerType is not an ID.
          if (!curr.validRValue())
            throw new GlimmerException("Cannot call func with invalid object:
"
                                       + curr.toString());
          current_table.add(params[i], curr);
        }

      control_stack.push(ControlType.RUN_FUNC);

        // Walk the statements in the function AST.
        ret = expr(function.getAST());

        // Restore the previous symbol table.
        current_table = old_table;

        // At the end of a function, if we exited via a return, then we need
to
        // reset to runnable status again (or whatever it was before).
      ControlType control;
      do {
        control = (ControlType)control_stack.pop();
      } while (control != ControlType.RUN_FUNC);
      }
    | #("load" load_image_name:ID load_file_name:STR
      {
        GlimmerImage image = new GlimmerImage();
        image.intrinsicLoad(load_file_name.toString());
        current_table.add(load_image_name.toString(), image);
      })
    | #("save" save_image_name:ID save_file_name:STR
      {
        first = current_table.get(save_image_name.toString());
        if (first == null || !(first instanceof GlimmerImage))
          throw new GlimmerException("Cannot save non-image object: "
                                     + save_image_name.toString());
        ((GlimmerImage)first).intrinsicWrite(save_file_name.toString());
      })
    | #("new" new_image_name:ID second=expr third=expr
      { if (second == null || third == null)
          throw new GlimmerException("Cannot create img w/o width and
height.");
        if (!(second instanceof GlimmerNum))
          throw new GlimmerException("Cannot create image w/ non-num width: "
                                     + second.toString());
        if (!(third instanceof GlimmerNum))
```

```
          throw new GlimmerException("Cannot create image w/ non-num height:
"
                                     + third.toString());
        GlimmerImage image = new GlimmerImage();
        image.intrinsicNew((int)(((GlimmerNum)second).getValue()),
                           (int)(((GlimmerNum)third).getValue()));
        current_table.add(new_image_name.toString(), image);
      })
    | #("print" first=expr
      { if (!first.validRValue())
          throw new GlimmerException("Cannot print invalid/undefined expr: "
                                     + first.toString());
        System.out.println(first.toString()); })
    | #("printstr" str:STR
      { System.out.println(str); })
    | #("break"
      { if ((ControlType)control_stack.peek() != ControlType.RUN_LOOP)
          throw new GlimmerException("Cannot break while not in loop.");
      control_stack.push(ControlType.BREAK); })
    | #("continue"
      { if ((ControlType)control_stack.peek() != ControlType.RUN_LOOP)
          throw new GlimmerException("Cannot continue while not in loop.");
      control_stack.push(ControlType.CONTINUE); })
    | #(num:NUM
       { try {
           ret = new GlimmerNum(Float.parseFloat(num.toString()));
         } catch (NumberFormatException e) {
           throw new GlimmerException(e.getMessage());
         }
       })
    | #(id:ID
      { ret = current_table.get(id.toString());
        if (ret == null) {
          ret = new GlimmerID();
          ((GlimmerID)ret).setID(id.toString());
        }
      })
  // Accessing pixel or image elements.
    | #("pixel_red"    first=expr  { ret = first.pixelRed();})
    | #("pixel_green"  first=expr  { ret = first.pixelGreen();})
    | #("pixel_blue"   first=expr  { ret = first.pixelBlue();})
    | #("pixel_alpha"  first=expr  { ret = first.pixelAlpha();})
    | #("pixel_x"      first=expr  { ret = first.pixelX();})
    | #("pixel_y"      first=expr  { ret = first.pixelY();})
    | #("image_height" first=expr  { ret = first.imageHeight();})
    | #("image_width"  first=expr  { ret = first.imageWidth();})
  // Referencing a pixel
    | #(LBRKT first=expr second=expr third=expr)
      {
        if (first == null ||
            !(first instanceof GlimmerPixel) &&
            !(first instanceof GlimmerImage))
          throw new GlimmerException("Cannot take pixel reference of " +
                                     "a non-pixel, non-image object.");
        if ( second == null || third == null ||
             !(second instanceof GlimmerNum) ||
             !(third instanceof GlimmerNum))
```

```
          throw new GlimmerException("Cannot take a non-num pixel
reference.");
        ret = first.pixelRef((int)(((GlimmerNum)second).getValue()),
                             (int)(((GlimmerNum)third).getValue()));
      }
  // Creating a pixel
    | #(PIXEL_CREATE first=expr second=expr third=expr fourth=expr)
      {
        if ( !(first instanceof GlimmerNum) ||
             !(second instanceof GlimmerNum) ||
             !(third instanceof GlimmerNum) ||
             !(fourth instanceof GlimmerNum))
          throw new GlimmerException("Cannot create pixels with non-num
args.");
        ret = new GlimmerPixel((int)(((GlimmerNum)first).getValue()),
                               (int)(((GlimmerNum)second).getValue()),
                               (int)(((GlimmerNum)third).getValue()),
                               (int)(((GlimmerNum)fourth).getValue()));
      }
  // Logical Operators
    | #(OR first=expr right_or:.)
      { if (!(first instanceof GlimmerNum)) {
          throw new GlimmerException("Cannot call logical-or on non-num
obj");
        }
        if (((GlimmerNum)first).isTrue()) {
          ret = GlimmerNum.TRUE_VALUE;
          break;
        }
        GlimmerType right = expr(#right_or);
        if (!(right instanceof GlimmerNum)) {
          throw new GlimmerException("Cannot call logical-or on non-num
obj");
        }
        if (((GlimmerNum)right).isTrue()) {
          ret = GlimmerNum.TRUE_VALUE;
          break;
        }
        ret = GlimmerNum.FALSE_VALUE;
      }
    | #(AND first=expr right_and:.)
      { if (!(first instanceof GlimmerNum)) {
          throw new GlimmerException("Cannot call logical-and on non-num
obj");
        }
        if (!((GlimmerNum)first).isTrue()) {
          ret = GlimmerNum.FALSE_VALUE;
          break;
        }
        GlimmerType right = expr(#right_and);
        if (!(right instanceof GlimmerNum)) {
          throw new GlimmerException("Cannot call logical-and on non-num
obj");
        }
        if (((GlimmerNum)right).isTrue()) {
          ret = GlimmerNum.TRUE_VALUE;
          break;
```

```
        }
        ret = GlimmerNum.FALSE_VALUE;
      }
    | #(NOT first=expr)                    { ret = first.not(); }
    | #(GE first=expr second=expr)         { ret = first.ge(second); }
    | #(LE first=expr second=expr)         { ret = first.le(second); }
    | #(GT first=expr second=expr)         { ret = first.gt(second); }
    | #(LT first=expr second=expr)         { ret = first.lt(second); }
    | #(EQ first=expr second=expr)         { ret = first.eq(second); }
    | #(NE first=expr second=expr)         { ret = first.ne(second); }
  // Mathematical operators
    | #(PLUS first=expr second=expr)       { ret = first.add(second); }
    | #(MINUS first=expr second=expr)      { ret = first.sub(second); }
    | #(TIMES first=expr second=expr)      { ret = first.mul(second); }
    | #(DIV first=expr second=expr)        { ret = first.div(second); }
    | #(MOD first=expr second=expr)        { ret = first.mod(second); }
  // Unary Operators
    | #(UMINUS first=expr)          { ret = first.neg(); }
    | #(UPLUS first=expr)
      { ret = new GlimmerNum(((GlimmerNum)first).getValue()); }
;

arg_list returns [ Vector<GlimmerType> args ] throws GlimmerException
{
  args = new Vector<GlimmerType>();
}
    : #(FUNC_ARGS
         (arg:. { args.add(expr(#arg)); })* )
;
```