C I L Y

Final Report

Jiwan **C**hoi
Edward **I**shak
Jonathan **L**iu
Makiko **Y**asui

Table of Contents

Chapter 1

Introduction

1.1 Overview of CILY

In the computer industry's recent progression, there has been a shift from a text-based world to a graphical-based one. Users would rather see pictures, images, and fancy graphics rather than just textual information. Until recently, manipulating images was a very expensive operation. It required a lot of space, computer time, and even user time. Graphics artists today still have to invoke an interactive application, such as Adobe Photoshop, in order to change colors, brightness, scale, etc.

The Choi Ishak Lui Yasui (CILY – pronounced "silly") Programming Language is designed to give the user the ability to programmatically manipulate 2D images. The language gives the user flexibility to perform a broad range of procedures by supporting large scale batch operations and also allowing the programmer to automate these processes. This dramatically saves valuable user time and allows quick preparation of images for a wide range of uses. CILY even provides an automated set of procedures to dynamically produce web sites for image viewing.

CILY is a safe programming language, automatically taking care of memory management and garbage collection. It is robust, scalable, and very intuitive to use. Overall, CILY will dramatically increase user productivity and programmers world-wide will be asking, "Why didn't we have this available before?"

1.2 Background / Related Work

As electronic services, such as e-mail and e-commerce, became vital parts of our lives, the World Wide Web has shifted from a text-based user interface to a graphics-based one. Websites with graphical components help users navigate and visualize their needs and desires effortlessly. Consequently, the computer industry has produced many applications for image manipulation and website development. Adobe Photoshop and Macromedia Dreamweaver are two commonly used applications that are widely used by web developers around the world.

Photoshop's primary tools are the image manipulation options it offers. In web development today, image manipulation, such as a simple resizing or a more complex brightness/contrast control, is an essential part of creating user-friendly and visually pleasing interfaces. Furthermore, it is very likely that mass image manipulation is needed to create a website. Photoshop provides this mass image manipulation functionality called "batch processing." Using "batch processing," the user can record a series of actions performed on one image, and subsequently apply the same exact series of actions on a directory of images.

Macromedia's Dreamweaver is an application that generates actual HTML code for a website. Dreamweaver made it possible for all users, even those with little knowledge of HTML, to create a website without any difficulty by making the interface graphical. At the same time, it allows advanced users to hardcode HTML. Similar to Photoshop, Dreamweaver allows mass file manipulations. For example, the users can create a "template page" and apply the template on numerous pages to put these files in the same format.

Although these two applications and many others help web developers in mass image manipulation and mass HTML file manipulation, there are limitations these processes. Once the user decides on the set of actions for batch processing, or a template for a web page, the user cannot add extra specifications according to the properties of each image, or each web page. CILY combines the basic functionalities of these two applications without these limitations.

1.3 Motivations for the Language

The users of CILY will appreciate the rich set of features offered in the language over other interactive image manipulation applications, such as Photoshop, Dreamweaver, and PhotoEditor, because it combines many of their functionalities and creates a new tool that extends image manipulation and web development in a new direction. The function calls and primitive data types are intuitive yet powerful, allowing both novice and advanced programmers to easily customize image manipulation procedures based on a set of parameters. The automation of performing these highly customized operations on a large set of images allows users to spend less time on repetitive tasks, such as opening and saving an image file, and to spend more time on developing the set of operations to be executed. Furthermore, by integrating image manipulation and webpage creation into one language, CILY simplifies the process of creating an image gallery on the web.

1.4 Goals of the Language

The goal of CILY is to allow users to combine the key features of Photoshop and Dreamweaver into one programmatically defined environment. It gives the users an easy to way to perform image manipulation procedures on a large set of images while giving the users more flexibility and power by specifying parameters for custom manipulation. These manipulative tasks can affect images at a level as low as the pixels of one image to as high as a directory of images. CILY will save enormous amounts of time for web developers by allowing mass manipulation in just few lines of code.

1.5 Features

1.5.1 Simple

CILY closely follows the widely used syntax of such procedural languages as Java, C, and C++, with added features that provide a more comprehensible grammar to users of image manipulation applications. For-each loops have been added to the language, while unnecessary features such as while loops and case structures have been omitted. CILY also facilitates the coding process by offering a set of real world object types such as CILYImage, CILYColor, and CILYPixel, as well as a set of intuitive function calls such as Rotate, SetColorAt, and ConvertToGreyScale. The underlying Java code automates the tasks of memory management and garbage collection, allowing users to focus on coding the image manipulation procedures rather than the lower level programming aspects. These automated tasks reduce the number of errors in the code, and consequently, reduce the implementation and debugging time for users.

1.5.2 Portable

CILY is designed to support a broad range of users as well as a variety of programming environments. Because CILY compiles to Java 1.4.1 source code, Java's portability features directly applies to CILY. Any environment supporting Java 1.4.1 Virtual Machine will allow the execution of CILY programs. Furthermore, these programs will not depend on implementation or the domain. The sizes of the primitive data types are fixed, and the behavior of arithmetic operations are specified.

1.5.3 Safe and Robust

By using Java source code as a target code for CILY compilers, CILY eliminates costly programming errors by automating memory management and garbage collection, therefore users can code without having to worry about overwriting memory and corrupting data, which could easily happen in C or C++. All primitive data types are initialized when a variable is declared to both reduce

errors and to simplify the coding process. When necessary, integers are automatically cast into floating points, however to avoid any side effects due to simple mistakes, no other primitive data type is converted into another type automatically. If the user mistakenly uses the wrong data type, the compiler would perform a type check, specifying the exact location of the type mismatch.

1.5.4 Time Efficient

The major advantage of using CILY over Dreamweaver, Photoshop, or PhotoEditor lies in the ability to customize procedures for a specific subset of images using Boolean operations. Unlike Photoshop, where a batch procedure is applied to all images in a specific directory, CILY allows each image in a directory to be altered differently within a single program. Thus, CILY greatly reduces the time required to manipulate a large quantity of images. With a few lines of CILY code, users can create a program to perform low-level tasks such as identifying pixels of certain RGB values, as well as such high-level tasks as opening and saving an image file, creating thumbnails from a set of images, and creating a webpage from the thumbnails. The time required to write and debug these programs is further reduced by the simplicity of the grammar and the automated memory management and garbage collection.

1.6 Model of Computation

Programs written in CILY will allow users to manipulate images at any level. Using real world object types, the user can procedurally manipulate images from a very low level (pixel RGB values) to a very high level (directory of images) using library function calls and a standard set of control-flow structures available in CILY. CILY uses basic control flow with the most widely used control-flow structures used in procedural languages today. This makes it easy and intuitive for the user to begin writing programs in CILY immediately without much training. The user is given quite a bit of flexibility, although not so much as to slice one's finger.

CILY supports basic I/O to allow users to write and read images to and from the disk. The details of I/O are abstracted from the user by high level function calls in order to perform basic operations. In addition, I/O operations that may ordinarily cause problems written in one language are performed safely and without side effects in CILY due to its safe environment programming environment.

Programs written in CILY will be compiled into Java source code. The Java Virtual Machine will then be used to execute the program. Using the Java Virtual Machine will not cause a problem because no animation or other special output is being created. As a side effect, images on disk may be manipulated if the user has programmatically specified this to happen.

1.7 Example Syntax

```
// Load Image
var Image i = Open("C:\images\oldImage.jpg");

// Invert Image
Invert(i);

// Save Image
Save(i);

// Load multiple images
Image[] i_array = OpenDir("C:\images\");

// Invert every image in the i_array
foreach img in i_array {

     Invert(i_array[img]);

}

// make a webpage with a specified title, option,
// input directory and output file
CreateAlbum("My Vacation", "jan, 3, 03", style, i_array);
```

Chapter 2

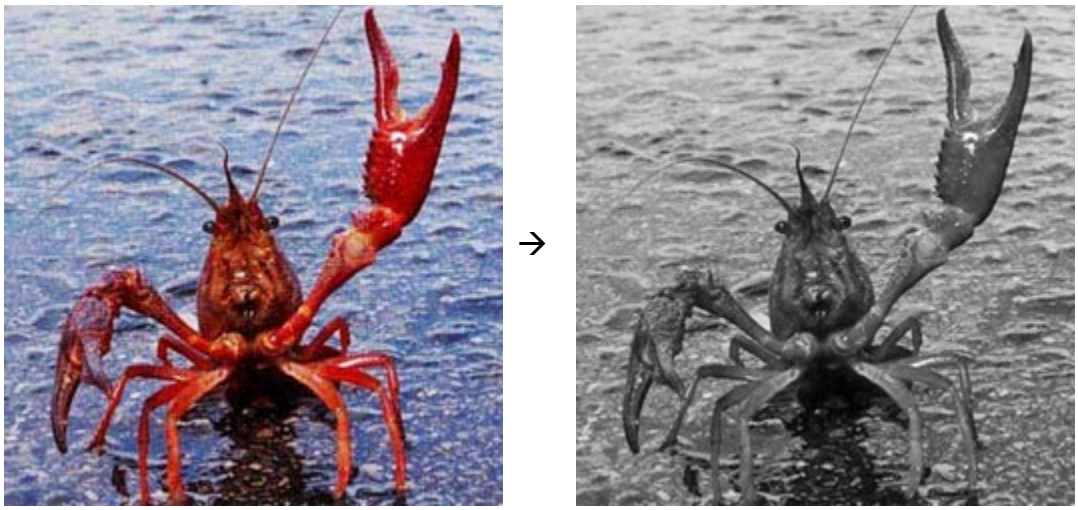
CILY Tutorial

2.1 Introduction

CILY is a functional language used for image manipulation. Some of the key features are the automation capabilities of the language and the webpage image gallery creator. The syntax and coding style mimic those of Java, C++, and C.

Remember, variable declarations must start with the reserved word "var."

2.2 First Example of Basic CILY Syntax



```
main()
{
        var int five = 5;      //single line comment; only kind allowed in CILY
        var double tenHalf = 10.5;
        var string newFileName = "myPictureGray.jpg";
        var boolean truth = true;
```

```
        var Image img = Open( “myPicture.jpg” );     //opens a picture for editing
        ConvertToGrayScale( img );                   //makes the image grayscale
        SaveAs( img, newFileName );          //saves the image to a new filename
}
```

Explanation of Basic CILY Syntax:

```
        var int five = 5;      //single line comment; only kind allowed in CILY
        var double tenHalf = 10.5;
        var string newFileName = “myPictureGray.jpg”;
        var boolean truth = true;
```

- This is an example of simple variable and comment syntax. These four assignments correlate to CILY’s four primitive types (`int, double, string, boolean`). Our four primitive types are a subset of Java’s type set. Comments in CILY are single line comments only, and are preceded by “//”.

```
        Var Image img = Open( “myPicture.jpg” );     //opens a picture for editing
        ConvertToGrayScale( img );                   //makes the image grayscale
        SaveAs( img, newFileName );          //saves image to new filename
```

- Here we open an image using the `Open` function and set it to type Image. We subsequently make it gray and save it again. For documentation on functions and function parameters, reference the CILY Language Reference Manual.

In this example, we created some variables and assigned them values. This is perhaps rather uninteresting. Beginning to work with images, we then proceeded to open an image, change it, and save it again under a new filename. The previous transformation only affected one image. What makes CILY unique is its ability to automate these transformations to directories of files: we will learn about it in the next step.

2.3 Automation Using foreach Loops

CILY tasks are automated by opening directories (instead of single files), using arrays to store the images of a directory, and using loops to make changes to each image in the array. Automated tasks can be as simple as renaming files to changing the whole look of each image based on its previous characteristics.

Example of Basic automation with the `foreach` loop:

```
main()
{
        //opens a directory of images, makes each file an Image & moves to array
        array Image[] myImages = OpenDir( “c:/cily/” );

        //foreach loop syntax… note: currentImage is of type int
        foreach currentImage in myImages
        {
                //blur the image
                Blur( myImages[currentImage] );
```

```
        }

        //save all images in array to new directory.
        SaveAll ( myImages );
}
```

Explanation of basic foreach automation:

`array Image[] myImages = OpenDir( “c:/cily/” );`

- The left-hand side of this statement creates an array called myImages. The right-hand side opens a directory of images, specifically c:/cily/. The `OpenDir` function makes each file in that directory an `Image` type and puts it in the left-hand side array myImages.
  Note: all array declarations must begin with the keyword `array`.

`foreach currentImage in myImages`

- This is CILY standard `foreach` loop declaration syntax. The declaration starts with the keyword `foreach`, and it is followed by an identifier (in this case currentImage). This identifier is the iterator: the current index number (an int) of the array myImages. The keyword `in` is simply to add clarity that currentImage is an index of the array myImages.
  Note: currentImage is of type `Image`

`Blur( myImages[currentImage] );`

- Having selected the area to transform, we can now apply a function to an image in the area of selection (`Blob`). In this example, we blur the image according to the `Blob` selection, which is the whole image in this case.

`SaveAll ( myImages );`

- Finally, after having changed the images, we can save all the images from the array.

2.4 CILY’s Webpage Automation

The last major feature of CILY is the webpage automator. This function can make a webpage of image thumbnails for you! It outputs an HTML file, and all it takes is one line of code.

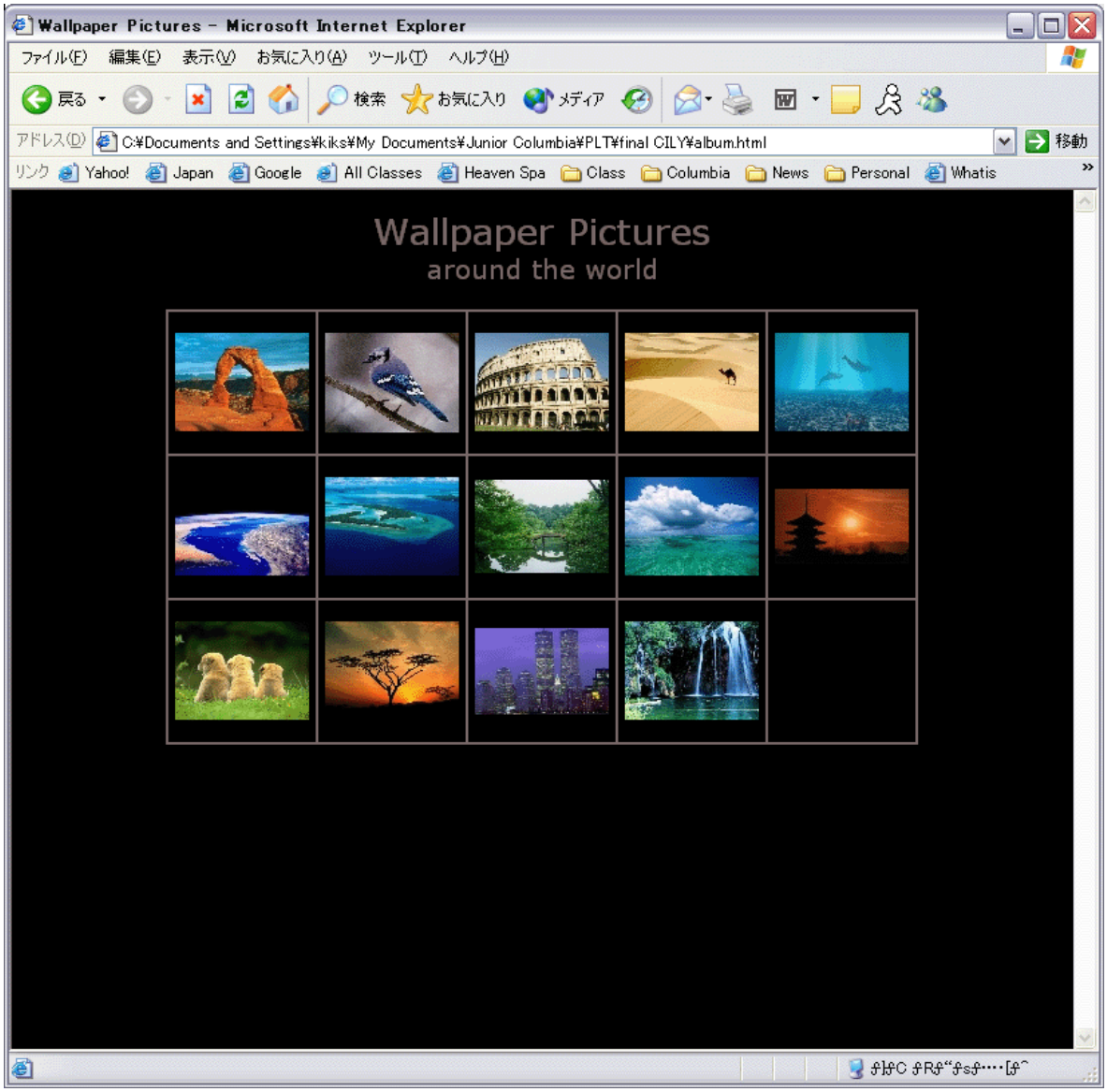
```
main()
{
        var string htmlFileName = “myHtmlFileName.html”;
        var string title = “My Amazing Photo Album”;
        var string subtitle = “photos of cool CS students”;
        var int styleNumber = 3;
        var Color bgcolor = Color(75,75,75);
        var Color fgcolor = Color(0, 0, 0);
        array Image[] pics = OpenDir( pathname );

        CreatePhotoAlbum( htmlFileName, title, subtitle, bgcolor, fgcolor,
styleNumber, pics);
```

```
}
```

Simply put, we have condensed the webpage photo album automation into one line of code using the `CreatePhotoAlbum` function. The parameter specifications can be found in the CILY LRM. Also, the input picture files for the Webpage Automation are in an array (pics) of Images; so this means that they can be from different directories and from different places, as long as they end up in the array before you call the CreatePhotoAlbum function.

Example HTML output:



Chapter 3

Language Reference Manual

1 Introduction

The CILY programming language is a simple, function-oriented image manipulation language that is based on and extends the popular Java programming language. CILY offers beginner and advanced programmers a text-based image manipulation tool, versus a graphical interface-based tool, that can perform large-scale batch operations and functions easily using only few lines of code. This language intends to be a useful tool for web developers and those who work with graphics. One of the major features is its webpage creator that creates a thumbnail photo album in HTML.

CILY is a relatively high level language that models its syntax and style after languages like C, C++, and Java. A major advantage of using CILY over popular programs such as Photoshop or Dreamweaver is the fact that CILY automates many simple tasks to eliminate users giving repetitive commands. Automation of image manipulation processes in batch operations and simplification of directory to webpage translations make this programming language an attractive alternative to mainstream executables.

CILY is loosely based on the Java programming language, and the CILY compiler uses Java source code as its target code. By using Java in this way, CILY can utilize Java's automated memory allocation and garbage collection making programming in CILY even easier.

2 Syntax Notation (of this manual)

`Lucida console` is used for literals and keywords.

Any (?.?) notation is a reference to another section. See that section for further details.

3 Lexical Convention

3.1 Programs

A CILY program consists of one source file. A source file is an ordered sequence of Unicode characters.

A program is compiled using five steps:

1. **Transliteration** – converts a file from a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. **Lexical analysis** – translates a stream of Unicode input characters into a stream of tokens.
3. **Syntactic analysis** – parses the stream of tokens to create an abstract syntax tree of tokens.
4. **Semantic analysis** – processes the tree of tokens and analyzes its meanings.
5. **Code generation** – translates the processed tokens into an executable Java code.

3.2 Separators

There are three types of separators: white space, line terminators, and comments.

1. **White space** – White space is defined as any character with Unicode Zs (which includes the space character) as well as the horizontal tab character (U+0009).
2. **Line terminator** – The definition of lines determines the line numbers produced by CILY compiler. Lines are terminated by the ASCII characters CR (also know as "return"), or LF (also known as "newline"), or CR LF.
3. **Comment** - The characters "//" introduce a single line comment. Only single line comments are permitted.

3.3 Types of tokens

3.3.1 Identifiers

An identifier is a sequence of letters, digits and "_". The first character must be a letter, and upper and lower case letters are considered different.

3.3.2 Keywords

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

```
Array           main
boolean         Oval
Color           Pixel
const           Point
double          print
else            println
```

```
elseif          Rectangle
false           return
foreach         string
function        Triangle
Heart           true
if              var
Image           void
in              while
int
```

3.3.3 Constants

3.3.3.1 Integer constant

An integer constant is a sequence of digits.

3.3.3.2 Floating constant

A Floating constant consists of an integer part, a decimal point, and a fraction part. Both integer part and fraction part consist of a sequence of digits. Either one of integer part or fraction part may be missing, but not both.

3.3.3.3 String constant

A String constant is a sequence of characters surrounded by double quotes “”. The constant begins with the first occurrence of a “” and ends with the last occurrence of a “”. There may be nested double quotes inside a string constant.

3.3.4 Expression operators

The following tokens are operators: reserved and unoverloadable.

```
+     -     *     /     %     !     =
+=    -=    *=    /=    %=    !=    ==
>     <     >=    <=    ++    --    ->
&&    ||
```

4 Storage Class, Types

4.1 Storage Class

All storage classes are automatic.

4.2 Primitive types

There are four primitive types, and no explicit casting is allowed.

4.2.1 int

`int` is 32-bit signed integral type, and it is a sequence of digits.

integer → *SIGNOP? (DIGIT)+;*
SIGNOP → *'+' | '-' ;*
DIGIT → *'0'..'9';*

4.2.2 double

`double` is a Double-precision floating point type.

double → *integer*
*| SIGNOP? (DIGIT)+ DOT (DIGIT)**
| SIGNOP? (DIGIT) DOT (DIGIT)+;*
SIGNOP → *'+' | '-' ;*
DIGIT → *'0'..'9';*

4.2.3 string

`string` is a sequence of characters surrounded by double quotes..

string → *' " '! (~(' " ' | ' \n ') | (' " '! ' " '))* ' " '!;*

4.2.4 boolean

`boolean` is a Boolean type, and it is either true or false.

BOOLEANCONST → *"true" | "false";*

4.3 Built-in types

There are eight different built-in types, four entities and four shapes. While all primitive types begin with a lowercase letter, all built-in types begin with an uppercase letter. The four entities are `Point, Color, Pixel, and Image`. The four shapes are `Rectangle, Oval, Triangle, Heart.`

4.3.1 Point

`Point` is defined by two integers, "x" and "y", which represent a certain coordinate.

4.3.2 Color

`Color` is defined by three integers, "r", "g", and "b" that represent a certain color. "r" (red), "g" (green), and "b" (blue) define the color.

4.3.3 Image

`Image` consists of two integers and a string, "width", "height" and "filename". "width" and "height" specify the size of an image, whereas "filename" specifies the path to the image file.

4.3.4 Rectangle

`Rectangle` is defined by four integers, "x", "y", "width" and "height". "x" and "y" specify the upper-left corner of the rectangle, and "width" and "height" specify the size of the rectangle.

4.3.5 Oval

`Oval` is defined by four integers, "x", "y", "width" and "height". "x" and "y" specify the center of the oval, and "width" and "height" specify the horizontal radius and the vertical radius of the oval respectively.

4.4 Derived types

4.4.1 Arrays

An array is a list of objects, which must all be of the same type. An array may be either one-dimensional or two-dimensional.

4.4.2 Functions

A function can return any type or an array of any type. Function definitions are further described in section 10.1

5 Variables

Variables are locations for storing information. The information stored in the variable is defined by the type that the variable is associated with.

Variables must be assigned some value before the variable contents can be accessed. When variables are declared without an assignment of an initial value, a default value is assigned.

5.1 Variable Categories

5.1.1 Global Variables

Global variables must be defined in the outermost scope of the program. They are accessible then from all functions and all interior scopes throughout the program since CILY programs can only consist of one file.

5.1.2 Local Variables

Local variables are those variables that are declared inside a block that is not the outermost block. Hence, its lifetime is completely dependent on the block it is contained in. Local variables are also not initially assigned.

Examples of local variable lifetimes occur in a `foreach` statement. The iteration variable is a local variable. A new instance of that variable is created each time an iteration occurs. Similarly, a new instance is created anytime recursion occurs with some local variable on the interior of the recursive block.

5.1.3 Const Variables

Const variables are variables that are initialized with the keyword `const` preceding the type. Oftentimes constants are listed as const variables because const variables pertain to the entire program and its value cannot be changed.

5.1.4 Instance Variables

Instance variables are variables without the `const` keyword. These variables have a scope local to the nearest set of brackets that surround it. The initial value is the initial value standard to the type the variable is associated with.

5.2 Default values

Const variables and instance variables are both initialized to the associated type's default value.

6 Conversions

A conversion allows one type to be treated as a different type. Since CILY only has 4 primitive types: integer, double, string, and boolean, the only conversions possible are from integer to double and double to integer. Again, this is further limited because the double type will never be cast as an integer due to CILY's no explicit casting rule (4.2). Hence we consider only int to double implicit conversions.

6.1 Integer to Double Implicit Conversion

When converting an integer to a double, the integer is simply cast as a double. This situation arises when a double and an integer are both part of an arithmetic operation and their differences cannot be resolved. The integer will then be converted to a double to allow the operation to proceed. A conversion from integer to string and double to string will occur only when using the '==' and '!=' operators. Otherwise, implicit conversion from a non-string type to a string would not occur.

6.2 Explicit Conversions

Explicit conversions are not allowed in CILY (4.2).

7 Expressions

There are eight types of expression operators in CILY. The precedence of these operators follows the order in which the operators appear in the following subsections. The operators with the highest precedence are discussed before those of lower precedence.

7.1 Left value primary expressions

Left value primary expressions group left to right, except for identifiers, which stand on their own. Left value primary expressions may appear on the either the left or the right side of an operator. Left value primary expressions will appear as *lval* in the sections following this section.

7.1.1 Identifier

Identifier is a reference to a properly declared variable discussed in section 8.1. An identifier must be a letter followed by zero or more characters, digits, or underscores.

id → *LETTER (DIGIT | LETTER | '_')*;*
LETTER → *'A'..'Z' | 'a'..'z';*
DIGIT → *'0'..'9';*

7.1.2 Array element access

A primary expression referring to an array contains the information of the location of the first element of the array. Therefore, square brackets with an index number within the brackets are used to access individual elements. The expression, a[i] returns the *i*th element of the array.

arrayaccessexpr → *lval '['expression ']';*

7.1.3 Type member access

A primary expression followed by a period and a type member name is also a primary expression. The return type is the type of the member. Thus, accessing the width of an Image type expression would return an int. However, the width and height of an image may not be changed using type member access. All other type members may be assigned a value using this
expression.

memberaccessexpr → *lval ARROW id;*
ARROW → *'->';*

7.2 Right value primary expressions

Right value primary expressions group left to right, except for constants, which stand on their own. Right value primary expressions may only appear on the right side of an operator, therefore, their values cannot be changed.

7.2.1 Constant

A constant is an integer, a floating-point number, a set of characters within double quotes, or a boolean constant. The four types of constants are `int, double, string, and boolean` respectively. A boolean constant is defined by the keywords, "`true`" or "`false`."

constant → *integer | double | string | BOOLEANCONST;*
integer → *(SIGNOP)? (DIGIT)+;*
double → *integer*
*| (SIGNOP)? (DIGIT)+ DOT (DIGIT)**
| (SIGNOP)? (DIGIT) DOT (DIGIT)+;*
string → *'"'! (~ ('"' | '\n') | ('"'! '"'))* '"'!;*
BOOLEANCONST → *"true" | "false";*

7.2.2 Function calls

A function call is a properly declared function name followed by a set of parentheses with possible list of arguments. The return value and type are defined in the function declaration discussed in section 10.1.

functioncall → *id '(' paramlist ')';*
paramlist → *expression (',' expression)**
*| /*empty*/;*

7.2.3 Parenthesized expression

A parenthesized expression does not change the value or the type of the expression within the parentheses.

parenexpr → *'(' expression ')';*

7.2.4 Arraylist

An arraylist is an explicit definition of an array that returns an array of the type defined by the elements in the arraylist. An arraylist containing all integers would therefore return an array of type int. Within a set of brackets, the elements of the array are listed in parentheses, separated by commas. Multidimensional arrays would require multiple parenthesized lists of elements.

arraylist → '{' (*arraylist** | *expression* (',' *expression*)*) '}';

7.3 Unary operators

Unary operators could be divided into unary evaluation operators and unary assignment operators. Unary evaluation operators (unary sign and unary logical negation operators) group right to left, while unary assignment operators (postfix increment and decrement operators) group left to right.

7.3.1 Unary sign operator

A minus or plus sign followed by an expression returns a value of the same type. The minus sign would return a value of the expression multiplied by -1, while a plus sign would not change the value. The expression that follows the operator must be of type `int` or `double`.

unarysignop → *SIGNOP expression;*
SIGNOP → *'+' | '-';*

7.3.2 Logical negation operator

The logical negation operator followed by an expression returns a `boolean` value opposite that of the expression. The expression that follows the operator must be of type `boolean`.

logicalnegop → *'!' expression;*

7.3.3 Postfix increment operator

The operator preceded by an expression returns the expression of the same value and type, then increments the value of the expression by one. The expression preceding the operator must be of type `int`.

incrementop → *lval "++";*

7.3.4 Postfix decrement operator

The operator preceded by an expression returns the expression of the same value and type, then decrements the value of the expression by one. The expression preceding the operator must of type `int`.

decrementop → *lval "--";*

7.4 Arithmetic operators

Arithmetic operators group left to right. These operators can be applied only to expressions of type other int or double, except for the remainder operator, which can be applied only to expressions of type `int`.

7.4.1 Multiplication Operator

This operator returns the product of the values of the expression preceding and following the operator. The expressions must be of type `int` or `double`. If at least one of the

expressions is of type `double`, the return type is `double`, otherwise, the return type is `int`.

multiplicationop → *expression '*' expression;*

7.4.2 Division operator

This operator returns the quotient of the values of the expression preceding and following the operator. The expressions must be of type `int` or `double`. If at least one of the expressions is of type `double`, the return type is `double`, otherwise, the return type is `int`.

divisionop → *expression '/' expression;*

7.4.3 Remainder operator (Modulus)

This operator returns the remainder after dividing the value of the expression preceding the operator by the value of the expression following the operator. Both expressions must be of type `int` or `double`. If they are of type `double`, the return value is the remainder of the truncated values of the expressions. The return type is always `int`.

remainderop → *expression '%' expression;*

7.4.4 Addition operator

The operator returns the sum of the values of the expression preceding and following the operator. The expression must be of type `int`, `double`, or `string`. The sum of two expressions of type `int` returns an `int`. The sum of two expressions of type `double` returns a `double`, and if one is of type `int` and the other is of type `double`, the return type is a `double`. If one of the expressions is of type `string`, the other must also be of type `string`. The sum of the string is the concatenated string, where the second string immediately follows the first string.

additionop → *expression '+' expression;*

7.4.5 Subtraction operator

The operator returns the difference of the values of the expression preceding and following the operator. The expressions must be of type `int` or `double`. If one of the expressions is of type `double`, the return type is `double`, otherwise, the return type is `int`.

subtractionop → *expression '-'expression;*

7.5 Relational operators

Relational operators group left to right, and the return type is always boolean. The expressions preceding and following these operators must be of type `int` or `double`.

7.5.1 Less-than operator

The operator returns `true` if the value of the expression preceding the operator is less than the value of the expression following it, otherwise, the return value is `false`.

lessthanop → *expression '<' expression;*

7.5.2 Greater-than operator

The operator returns `true` if the value of the expression preceding the operator is greater than the value of the expression following it, otherwise, the return value is `false`.

greaterthanop → *expression '>' expression;*

7.5.3 Less-than-or-equal-to operator

The operator returns `true` if the value of the expression preceding the operator is less than or equal to the value of the expression following it, otherwise, the return value is `false`.

lessthanequalop → *expression '<=' expression;*

7.5.4 Greater-than-or-equal-to operator

The operator returns `true` if the value of the expression preceding the operator is greater than or equal to the value of the expression following it, otherwise, the return value is `false`.

greaterthanequalop → *expression '>=' expression;*

7.6 Equality operators

Equality operators group left to right, and the return type is always `boolean`. The expressions preceding and following these operators can be of any type as long as they are of the same type.

7.6.1 Equal operator

The operator returns true if the value of the expression preceding the operator is equal to the value of the expression following it, otherwise, the return value is false. If the type of both expressions is not `int`, `double`, `string`, or `boolean`, the values of the members are checked. For example, comparing a string value "10" to an integer value 10 will return true.

equalop → *expression '==' expression;*

7.6.2 Not-equal operator

The operator returns true if the value of the expression preceding the operator is not equal to the value of the expression following it, otherwise, the return value is false. If the type

of both expressions is not `int, double, string,` or `boolean`, the values of the members are checked.

notequalop → *expression '!=' expression;*

7.7 Logical operators

Logical operators group left to right. The return type is always of type `boolean`.

7.7.1 And operator

The operator returns `true` if the values of the expressions preceding and following the operator both return either a positive number or `true`, otherwise, the return value is `false`.

andop → *expression '&&' expression;*

7.7.2 Or operator

The operator returns `true` if at least one of the values of the expressions preceding and following the operator returns either a positive number or `true`, otherwise, the return value is `false`.

orop → *expression '||' expression;*

7.8 Assignment operators

Assignment operators group left to right, unless they are unary assignment operators discussed in section 7.2.2 and 7.2.3.

7.8.1 Simple assignment operator

The operator replaces the value of the identifier or its member preceding the operator by the value of the expression following the operator. The identifier and the expression must be of the same type. However, if the identifier is of type `int`, and the expression is of type `double`, the value of identifier is replaced by the truncated value of the second. If the identifier is of type `double`, and the expression is of type `int`, the value of the second expression is converted to `double` and is assigned to the identifier. If identifier is an array, the expression must return an array.

simpleassignop → *lval '=' expression;*

7.8.2 Multiply-equal operator

multiplyequalop → *lval '*=' expression;*

7.8.3 Divide-equal operator

divideequalop → *lval '/=' expression;*

7.8.4 Mod-equal operator

modequalop → *lval '%=' expression;*

7.8.5 Plus-equal operator

plusequalop → *lval '+=' expression;*

7.8.6 Minus-equal operator

minusequalop → *lval '-=' expression;*

The operator replaces the value of the identifier or its member preceding the operator with the value of (*id or member of id) OP expression*, where OP is the operator preceding the equal sign, described in section 7.3.

7.9 Constant expressions

Constant expressions are formed by applying arithmetic operators to constants. They must evaluate to a constant after the operations are performed. Constant expressions may evaluate to `int, double,` or `string`. Thus, 1+1-1 returns the value 1 of type `int`, while "ab" + "cd" returns the value "abcd" of type `string`.

8 Declarations

Declarator expressions are used to specify the type of each identifier. The types can be of any type discussed in section 4. The value of the identifier may or may not be assigned during the declaration. Only one variable, array, or constant may be declared and/or assigned in each declaration.

8.1 Variable declaration

Only one variable may be declared in each declaration.

vardeclaration → *"var" TYPE id ('=' expression)?;*

8.2 Array declaration

arraydeclaration → *"array" TYPE*
(('[' expression ']' ('[' expression ']')?) id
| ('[' ']' ('[' ']')?) id ('=' (arraylist | expression))?);

An array declaration must follow the keyword "array," followed by the type and at least one set of square brackets. The square brackets may or may not contain an expression that evaluates to an integer specifying the size of the array. For multidimensional arrays, the size of all dimensions must be specified or none must be specified. If an assignment expression follows the declaration, the size of the array must not be specified. If the size is not specified, and an assignment expression does not follow the declaration, the constant 1 is used for the size. Only one array, one-dimensional or multidimensional, may be declared in each declaration.

8.3 Const declaration

constdeclaration → *"const" TYPE id '=' expression*

A const declaration specifies the type and the value of an identifier. Once it is declared, the identifier's value remains fixed throughout the program. Any attempt to assign a new value will result in an error. The declaration must follow the keyword "const," followed by a variable declaration.

9 Blocks and Statements

Statements are executed sequentially, unless otherwise indicated.

9.1 Declaration statements

declarationstmt → *vardeclaration* | *constdeclaration* | *arraydeclaration ';' ;*

A declaration statement is a single statement that declares a variable. Declarations are discussed in section 8.

9.2 Assignment statements

assignmentstmt → *lval ASSIGNOP expression ';' ;*
ASSIGNOP → *'=' | "+=" | "-=" | "*=" | "/=" | "%=";*

An assignment statement is a statement that assigns a value to a variable or a member of a variable.

9.3 Functioncall statements

functioncallstmt → *functioncall ';' ;*

(expression | declaration) ';' ;

A functioncall statement is a single function call. Function calls are discussed in section 7.

9.4 Selection statement

ifelsestmt → *"if" '(' expression ')' '{' block '}'*
*("elseif" '(' expression ')' '{' block '}')**
("else" '{' block '}')? ;

A selection statement executes a set of expressions within a block that satisfies a certain condition. If the expression returns the boolean `true`, the corresponding block is executed, and the other cases are ignored. If all the cases fail to match, the block corresponding to the "else" case is executed, or there is no "else" case, then the statement following the selection statement is executed.

9.5 Iteration statement

In one iteration, the body of the iteration statement is executed repeatedly as long as certain conditions hold true.

9.5.1 While statement

whilestmt → *"while" '(' expression ')' '{' block '}' ;*

A while statement is executed until the expression following the "`while`" keyword returns false. The expression must always return a `boolean` and is evaluated before the execution of the block.

9.5.2 Foreach statement

foreach → *"foreach" id "in" expression '{' block '}' ;*

A `foreach` statement executes the block as many times as the number of elements in a given array, keeping a counter for the current iteration. The expression is the same as using a while loop and looping until the end of the array.

9.6 Return statements

returnstmt → *"return" (expressionstmt)? ';' ;*

Return statement breaks out of the function and returns to the position where the function was called. If the function returns a value, the expression statement following the "return" must of the same type as the return type of the function. For all functions, there must be only one return statement, which must come at the end of the function body.

9.7 Blocks

block → *(statement)*;*

A block consists of a list of variable declaration, array declaration, and statements. A const declaration may not be inside of a block. If the block is the body of a function that returns a void, the return statement may be omitted, but otherwise, the return statement marks the end of the execution of a block.

10 Functions

10.1 Function Definition

A CILY program can consist of function definitions. Function definitions must appear *before* (in the code: above) the block of code where the function is called. Functions have the form:

function-definition:
return-type function-declarator function-body

10.1.1 Function Declarator

The function-declarator is defined by the following:

function-declarator:
declarator (*parameter-list* $_{opt}$)

parameter-list:
type identifier
| *type identifier, parameter-list*

10.1.2 Function Body

The function-body is defined by the following:

function-body:
{ *type-declarations function-statements returnstmt* }

Here is a simple example of a complete function definition:

```
function int maxPixelRValue ( Pixel p1, Pixel p2 )
{
      var int redValue;
      if ( p1.color.red > p2.color.red )
      {
            redValue = p1;
      }
      else
      {
            redValue = p2;
      }
      return redValue;
}
```

Here, *int* is the return type. This specifies that this function must return an integer value at the end of the function body.

The identifier *maxPixelRValue* is the name of the function. Programmers who wish to use this function should refer to it by this name.

The parameter list has two items, both *Pixel* objects referenced by the names *p1* and *p2*, respectively. The scope of these objects is bounded by the function body block.

The function body is surrounded by mandatory curly braces and every body must end with a return statement. For a function with a return type of "void," a return statement must not return any value.

10.2 Library Functions

10.2.1 Basic Operations Functions

10.2.1.1 Open (path)

prototype: Image Open (string path)
description: Opens an image file from the specified path and returns the contents of the image as an Image object. If the file does not exist, is empty, or corrupt, null is returned.
params: path (string) – the absolute or relative path of the image file to open
returns: the image to be opened if successful, null otherwise.

10.2.1.2 OpenDir (dir_path)

prototype: Image[] OpenDir (string dir_path)
description: Opens an entire directory of images in the given dir_path. Returns an array of Image objects where each element is an .gif image in the given path. Empty, corrupt image files, or files of other types than .gif are not returned. If there are no images in the given path, empty array is returned.
params: dir_path (string) – the absolute or relative path of the directory of images to open
returns: the array of images in the specified directory if successful, empty array otherwise.

10.2.1.3 Save (img)

prototype: void Save (Image img)
description: saves the image with its original name.
params: img (Image) – the image to save
returns: none

10.2.1.4 SaveAs (img, path)

prototype: void SaveAs (Image img, string path)
description: saves the image to a file specified as path.
params: img (Image) – the image to save
path (string) - the absolute or relative path of the image to be saved
returns: none

10.2.1.5 Save All(imgs)

prototype: void Save (Image[] imgs)
description: saves an array of images.

params: imgs (Image[]) – the array of images to save
returns: none

10.2.1.6 Copy (img1, img2)
prototype: void Copy (Image img1, Image img2)
description: copies the specified source image to destination image.
params: img1 (Image) – the copy of srcImg
img2 (Image) – the image to copy
returns: none

10.2.1.7 Paste (img1, img2, x, y)
prototype: void Paste (Image img1, Image img2, int x, int y)
description: pastes img2 on top of img1 at the location x,y.
params: img1 (Image) – the destination image
img2 (Image) – the source image
x (int) – the x location on img1 to paste the top leftmost pixel of img2
y (int) – the y location on img1 to paste the top leftmost pixel of img2
returns: none

10.2.1.8 Paste (img, pixels)
prototype: void Paste (Image img, Pixel[] pixels)
description: replaces the pixels of img with the pixels in the array.
params: img (Image) – the destination image
pixels (Pixel[]) – the array of pixels
returns: none

10.2.1.9 Crop (img, x, y, w, h)
prototype: void Crop (Image img, int x, int y, int w, int h)
description: crops img at the location x, y with width w and height h.
params: img (Image) – the image to crop
x (int) – the x location on img to crop
y (int) – the y location on img to crop
w (int) – the width of the cropping area
h (int) – the height of the cropping area
returns: none

10.2.1.10 Crop (img, rect)
prototype: void Crop (Image img, Rectangle rect)
description: crops img to the area of the specific rectangle
params: img (Image) – the image to crop
rect (Rectangle) – the rectangle area to crop
returns: none

10.2.1.11 Rotate (img, degree)
prototype: void Rotate(Image img, int degree)
description: rotates the image to specified degree
params: img (Image) – the image to rotate
degree (int) – degree to rotate
returns: none

10.2.1.12 FlipHorizontal(img)
prototype: void FlipHorizontal(Image img)
description: flips the image horizontally
params: img (Image) – the image to flip horizontally
returns: none

10.2.1.13 FlipVertical (img)
prototype: void FlipVertical (Image img)
description: flips the image vertically
params: img (Image) – the image to flip vertically
returns: none

10.2.1.14 Resize(img, newWidth, newHeight)
prototype: void Resize(Image img, int newWidth, int newHeight)
description: resizes the image based on width and height parameters. Constrains proportions.
params: img (Image) – the image to resize
newWidth (int) – the new maximum width for the image
newHeight (int) – the new maximum height for the image
returns: none

10.2.1.15 PasteTransparent(dstImg, srcImg, x, y, alpha)
prototype: void PasteTransparent(Image dstImg, Image srcImg, int x, int y, int alpha)
description: Pastes srcImg onto dstImage at location x,y using an alpha transparency.
params: dstImg (Image) – the image to paste onto
srcImg (Image) – the image to paste
x (int) – x location to paste
y (int) – y location to paste
alpha (int) – the transparency value (0 to 255)
returns: none

10.2.1.16 PasteTransparent(dstImg, pixels, alpha)
prototype: void PasteTransparent(Image dstImg, Pixel[] pixels, int alpha)
description: Pastes pixels onto dstImage at location x,y using an alpha transparency.
params: dstImg (Image) – the image to paste onto
pixels (Pixel[]) – the pixels to paste
alpha (int) – the transparency value (0 to 255)
returns: none

10.2.2 Color Manipulation Functions

10.2.2.1 Invert (img)
prototype: void Invert (Image img)
description: inverts the pixels of the entire image.
params: img (Image) – the image to be inverted
returns: none

10.2.2.2 Blur (img)
prototype: void Blur (Image img)
description: blurs the entire image
params: img (Image) – the image to apply the blur

returns: none

10.2.2.3 Brighten (img, degree)
prototype: void Brighten (Image img, int degree)
description: brightens the entire image to a certain degree.
params: img (Image) – the image to brighten
degree (int) – the degree of brightening to be applied. Values -100 to 100 accepted.
returns: none

10.2.2.4 Sharpen (img)
prototype: void Sharpen (Image img)
description: sharpens the entire image
params: img (Image) – the image to apply the sharpen
returns: none

10.2.2.5 DetectEdges (img)
prototype: void DetectEdges (Image img)
description: detect edges of the entire image
params: img (Image) – the image to apply the edge detector
returns: none

10.2.2.6 ConvertToGrayScale (img)
prototype: void ConvertToGrayScale (Image img)
description: makes all the pixels of the Image object grayscale.
params: img (Image) – the image to apply the grayscale to
returns: none

10.2.2.7 AddBorder (img , color, thickness)
prototype: void AddBorder (Image img, Color color, int thickness)
description: adds border to the image of specified color and thickness.
params: img (Image) – the image to add borders to
color (Color) – the color of the border
thickness (int) – the thickness of the border
returns: none

10.2.2.8 GetPixelAt (img, x, y)
prototype: Color GetPixelAt (Image img, int x, int y)
description: returns the pixel of the pixel at location x,y
params: img (Image) – the source image
x (int) – the x value of the pixel
y (int) – the y value of the pixel
returns: the Pixel of the pixel at x, y

10.2.2.9 GetColorAt (img, x, y)
prototype: Color GetColorAt (Image img, int x, int y)
description: returns the color of the pixel at location x,y
params: img (Image) – the source image
x (int) – the x value of the pixel whose color is to be returned
y (int) – the y value of the pixel whose color is to be returned
returns: the Color of the pixel at x, y

10.2.2.10 SetColorAt (img, color, x, y)
prototype: void SetColorAt (Image img, Color color, int x, int y)
description: changes the color at x,y to the specified color
params: img (Image) – the source image
color (Color) – the color to change the pixel to
x (int) – the x value of the pixel whose color is to be changed
y (int) – the y value of the pixel whose color is to be changed
returns: none

10.2.3 Selection Functions

10.2.3.1 SelectRect (img, rectangle)
prototype: Pixel[] SelectRect (Image img, Rectangle)
description: returns an array of pixels that lie inside the rectangle
params: img (Image) – the source image
rectangle – the selected rectangular area
returns: an array of Pixel objects

10.2.3.2 SelectOval (img, oval)
prototype: Pixel[] SelectOval (Image img, CILYOval)
description: returns an array of pixels that lie inside the oval
params: img (Image) – the source image
oval – the selected circular area
returns: an array of Pixel objects

10.2.3.3 MagicWand (img, color, tolerance)
prototype: Pixel[] MagicWand (Image img, Color, int)
description: returns an array of pixels with RGB value within the tolerated offset from color
params: img (Image) – the source image
color – the base color that will be compared
tolerance – the offset from the base color
returns: an array of Pixel objects

10.2.3.4 SelectInverse (img, pixels)
prototype: Pixel[] SelectInverse (Image img, Pixel[] pixels)
description: returns an array of pixels that are not included in the currently selected pixels
params: img (Image) – the source image
pixels – the selected array of pixels
returns: an array of Pixel objects

10.2.4 Webpage Creator Functions

`10.2.4.1 CreateAlbum ( filename, title, subtitle, bgcolor, fgcolor, imgs )`
prototype: void CreateAlbum(String title, String subtitle, String bgcolor, String fgcolor, Image[] imgs)
description: using all the images in the imgs array, creates a photo album webpage (album.html) using background color, foreground color. The filename, title and subtitle are also specified.
params: filename (string) – the name of the file to create

title (string) – the title of the photo album which will appear in large font
subtitle (string) – the subtitle of the photo album which will appear in med font
bgcolor (Color) – the background color of the photo album webpage
fgcolor (Color) – the background color of the photo album webpage
imgs (Image[]) – the array of images to compose the photo album

returns: none

10.2.4.2 CreateAlbum (filename, title, subtitle, bgcolor, fgcolor, numPics, thumbwidth, thumbheight, imgs)

prototype: void CreateAlbum(String filename, String title, String subtitle, String bgcolor, String fgcolor, int numPics, int thumbheight, int thumbwidth, Image[] imgs)

description: using all the images in the imgs array, creates a photo album webpage using the thumbnail dimensions, background color, foreground color. The filename, title and subtitle are also specified.

params: filename (string) – the name of the file to create
title (string) – the title of the photo album which will appear in large font
subtitle (string) – the subtitle of the photo album which will appear in med font
bgcolor (Color) – the background color of the photo album webpage
fgcolor (Color) – the background color of the photo album webpage
numPics (int) – the number of pictures in one row
thumbwidth (int) – the width of the thumbnails
thumbheight (int) – the height of the thumbnails
imgs (Image[]) – the array of images to compose the photo album

returns: none

11 Lexical Scope

The *scope* of a declaration is the region of the program within which the entity declared by the declaration can be referred to using a simple name. An entity is *in scope* at a particular point in a program if its declaration's scope includes that point.

Scopes can be nested and the inner scope can re-declare an entity of an outer scope. An entity is *hidden* if it is re-declared in an inner scope with the same name. The scoping rules for variables are designed to guarantee that the meaning of an entity's name used in an expression is always the same within that block or scope.

The scope of an entity generally includes the statement following through the end of the current block, including all nested blocks unless hidden by a nested entity with the same name. Blocks are typically defined by open and closed curly braces ('{' and '}').

The scope of a global entity includes the statement immediately following the declaration of that entity through the rest of the program.

The scope of a local entity declared within a function includes the line immediately following the declaration through the end of the function definition.

The scope of a function parameter is the entire function body

Appendix – CILY Grammar

```
protected DOT:          '.';
TILDE:                  '~';
NOT:                    '!';
PLUSOP:                 '+';
MINUSOP:                '-';
OPENPAREN:              '(';
CLOSEPAREN:             ')';
SEMICOLON:              ';';
OPENBRACE:              '{';
CLOSEBRACE:             '}';
OPENBRACKET:            '[';
CLOSEBRACKET:           ']';
COMMA:                  ',';

EQUALEQUALOP:           "==";
NOTEQUALOP:             "!=";
ASSIGNEQUALOP:          "=";

PLUSEQUALOP:            "+=";
MINUSEQUALOP:           "-=";
MULTEQUALOP:            "*=";
DIVEQUALOP:             "/=";
MODEQUALOP:             "%=";
PLUSPLUSOP:             "++";
MINUSMINUSOP:           "--";
ARROWOP:                "->";

protected
Digit:                  '0'..'9';

NewLine:                ('\n' | "\r\n" | '\r');

Whitespace:             (' ' | '\t' | '\f')+;

Id                      ('A'..'Z' | 'a'..'z') ('0'..'9' | 'A'..'Z' | 'a'..'z' | '_')*;

Comments:               "//" ((~'\n'))* '\n';

StringConstant:         '"'! ( ~('"' | '\n' ) | ('"'! '"' ))* '"'!;

RelationalOp:           '>' ('=')? | '<' ('=')?;

LogicalOp:              "&&" | "||";

ArithmeticOp:           ('*' | '/' | '%');


Number:                 (Digit)+ (DOT (Digit)*)? | DOT (Digit)+;
```

```
constant:           Number
                    | StringConstant
                    | booleanConstant
                    ;

booleanConstant:    "true"
                    | "false"
                    ;

varType:            "int"!
                    | "double"!
                    | "string"!
                    | "boolean"!
                    | "void"!
                    | "Color"!
                    | "Pixel"!
                    | "Image"!
                    | "Rectangle"!
                    | "Point"!
                    ;

declaration:        varType (OPENBRACKET CLOSEBRACKET
                    (OPENBRACKET CLOSEBRACKET)?)? Id
                    ;

arglist:            declaration (COMMA declaration)*
                    | // empty
                    ;

expression:         expression1 expression1tail
                    ;

expression1tail:    LogicalOp expression1 expression1tail
                    | // empty
                    ;

expression1:        expression2 expression2tail
                    ;

expression2tail:    (ASSIGNEQUALOP | PLUSEQUALOP |
                    MINUSEQUALOP | MULTEQUALOP | DIVEQUALOP |
                    MODEQUALOP ) expression2 expression2tail
                    | // empty
                    ;

expression2:        expression3 expression3tail
                    ;

expression3tail:    (RelationalOp | EQUALEQUALOP | NOTEQUALOP)
                    expression3 expression3tail
                    | // empty
```

```
                    ;

expression3:        expression4 expression4tail
                    ;

expression4tail:    (PLUSOP | MINUSOP) expression4 expression4tail
                    | // empty
                    ;

expression4:        expression5 expression5tail
                    ;

expression5tail:    ArithmeticOp expression5 expression5tail
                    | // empty
                    ;

expression5:        (PLUSOP | MINUSOP | NOT)?  rvalue
                    ;

rvalue:              lvalue
                    | OPENPAREN expression CLOSEPAREN
                    |  functioncall
                    | constant
                    ;

lvalue:             Id  lvaluetail
                    ;

lvaluetail:         (OPENBRACKET expression CLOSEBRACKET
                    (OPENBRACKET expression CLOSEBRACKET)?)
                    lvaluetailtail
                    | lvaluetailtail
                    ;

lvaluetailtail:     PLUSPLUSOP
                    | MINUSMINUSOP
                    | (ARROWOP Id lvaluetailtail)
                    | // empty
                    ;

paramlist:          expression (COMMA expression
                    | // empty
                    ;

statement:          whilestmt
                    | foreachstmt
                    | ifelsestmt
                    | functioncallstmt
                    | assignmentstmt
                    | printstmt
                    ;
```

printstmt: **"print" OPENPAREN expression CLOSEPAREN SEMICOLON**
|"println" OPENPAREN expression CLOSEPAREN SEMICOLON
;

whilestmt: **"while" OPENPAREN expression CLOSEPAREN OPENBRACE blockofstatements CLOSEBRACE**
;

foreachstmt: **"foreach" Id "in" lvalue OPENBRACE blockofstatements CLOSEBRACE**
;

ifelsestmt: **"if" OPENPAREN expression CLOSEPAREN OPENBRACE blockofstatements CLOSEBRACE ("elseif" OPENPAREN expression CLOSEPAREN OPENBRACE blockofstatements CLOSEBRACE)* ("else" OPENBRACE blockofstatements CLOSEBRACE)?**
;

functioncallstmt: **functioncall SEMICOLON**
;

functioncall: **(Id | varType) OPENPAREN paramlist CLOSEPAREN**
;

returnstmt: **"return" (expression)? SEMICOLON**
;

assignmentstmt: **lvalue ((ASSIGNEQUALOP | PLUSEQUALOP | MINUSEQUALOP | MULTEQUALOP | DIVEQUALOP | MODEQUALOP) expression)? SEMICOLON**
;

blockofstatements: **(arraydeclaration | vardeclaration | statement) blockofstatementstail**
| // empty
;

blockofstatementstail: **(SEMICOLON)? (arraydeclaration | vardeclaration | statement) blockofstatementstail**
| // empty
;

vardeclaration: **"var" varType Id (ASSIGNEQUALOP expression)? SEMICOLON**
;

arraylist: **OPENBRACE ((arraylist)* | expression (COMMA**

```
                    expression)*) CLOSEBRACE
                    ;

arraydeclaration:   "array" varType
                    ((OPENBRACKET expression CLOSEBRACKET
                    (OPENBRACKET expression CLOSEBRACKET)? Id)
                    | (OPENBRACKET CLOSEBRACKET (OPENBRACKET
                    CLOSEBRACKET)? Id
                    (ASSIGNEQUALOP (arraylist | expression))))
                    SEMICOLON
                    ;

constdeclaration:   "const" varType Id ASSIGNEQUALOP constant
                    SEMICOLON
                    ;

functiondeclaration: "function" varType (OPENBRACKET CLOSEBRACKET
                    (OPENBRACKET CLOSEBRACKET)?)? Id OPENPAREN
                    arglist CLOSEPAREN OPENBRACE blockofstatements
                    returnstmt CLOSEBRACE
                    ;

globaldeclaration:  (vardeclaration | constdeclaration | functiondeclaration |
                    arraydeclaration)+
                    ;

mainfunction:       "main" OPENPAREN CLOSEPAREN OPENBRACE!
                    blockofstatements CLOSEBRACE
                    ;

program:            (globaldeclaration)?  mainfunction
                    ;
```

Chapter 4

Project plan

4.1 Organization of the project processes

4.1.1 Planning

To plan ahead and hit the mini-deadlines, our CILY project group used frequent meetings to facilitate this process. Our group regularly met about three times a week (Tuesday, Friday, Sunday) in order to perform group work, schedule our coming weeks, and delegate tasks and deadlines. We also met bi-weekly with our TA, Michael Locasto, who helped us stay on track, stick to our schedule, and provide insight as to our next steps in developing our translator.

By keeping the lines of communication open over e-mail and during our meetings, we were able to keep mostly to our schedule, only falling behind about half a week because of final exams. By meeting multiple times per week, our group was able to easily recognize which parts of our project were falling behind and which parts were ahead of schedule. See project timeline and project log in sections 4.5 and 4.6

4.1.2 Developing

In our different stages of development, our project group implemented different strategies for completing tasks and developing modules. By splitting up the work as appropriate to the task proved helpful because everyone still knew what was going on with the project despite not working directly with each component.

In the beginning, the whole group met together to develop our language from scratch. The ensuing step, to write the grammar, was also taken on by the CILY group as a whole so that every member could familiarize themselves with the grammar constructs and the CILY language syntax and semantics.

After the grammar was completed, the workload became more divided as tasks more easily fell into different categories. The four main components were the Lexer/Parser/ASTParser/CodeGeneration, Testing, Functions, and Documentation. We did not split the work by giving one person each task, but

rather divvied up the work in pairs to try and maximize the quality of the work being done.

4.1.3 Testing

The testing process was performed by creating test cases, and then using an automation tool, developed by Eddie, to facilitate the testing process. We had testing procedures throughout the process and it was basically split into four segments: syntax tests, semantics tests, code generation tests, and then overall bug-fixing. The details of the test plan and the testing regiment is described in detail in Chapter 6.

4.2 Roles and responsibilities

Jiwan Choi	Image Manipulation Functions Basic Operation Functions Webpage Creator Functions Grammar / Lexer / Parser Documentation { Whitepaper, Language Reference Manual }
Eddie Ishak	Grammar / Lexer / Parser AST / Code Generator Testing { test cases, automation, regression testing } Powerpoint presentation Image Manipulation Functions Documentation { Langage Reference Manual, test plan }
Jonathan Liu	Documentation { Introduction, Project Plan, Language Tutorial, Architectural Design, Test Plan, Language Reference Manual } Webpage Creator Functions Testing { test cases } Grammar Group Leader – submitted online copies –
Makiko Yasui	Grammar / Lexer / Parser AST Parser / Code Generator Bug Fixing Image Manipulation Functions Documentation { Language Reference Manual, Architectural Design}

4.3 Programming Style Guide

4.3.1 Introduction to Coding Standards

This document serves as a basic standard of coding practices for the CILY project group during development of the translator. Besides having quality source code, one must also pay attention to the style of programming so as to provide a third

party some readability. This includes the location and use of curly braces, tabs, white space, and semicolons. Although perhaps difficult due to the vast amount of different programming styles, the CILY project group members should attempt, as much as possible, to adhere to these standards in their code.

4.3.2 ANTLR Style Guide

In our ANTLR-compatible grammar, our style convention was to vertically align the rules so that each "or" line would fall directly under the one above it. The semi-colon rests at the end of the rule on its own new line.

```
style:     foo
           | bar
           | something
           ;
```

4.3.3 Java Style Guide (and for all use of curly braces)

For this project, we chose to align both curly braces on the left side of the block. Any block statement should have the two curly braces one directly on top of the other, and obviously at the beginning and end of the block. Also, the contents of the block should be one tab to the right of the curly braces that the block content is most closely a part of.

```
Block statement( )
{
     one tab here;
}
```

Comments are either single line or multiple line. Clearly mark your comments and they should be helpful in trying to understanding the code better.

```
// comments are clearly marked
/* multi-line comments say something
     meaningful */
```

4.4 Software Development Environment

This project was not developed on, or for, any specific operating system. Since we compile into Java code, the output files are very portable and can be used on any system that can run Java files. The grammar, lexer, parser, and tree parser were all developed using ANTLR, the suggested tool, since many of the processes are automated once the grammar is free of ambiguity and recursion. ANTLR produces Java code for us at this point. The version of ANTLR we used is v. 2.7.2. For the code generation and the library functions (image manipulation), we

used Java to develop these modules. The version of Java we were running is v. 1.4.02. More specifically, for the image manipulation library functions, we mainly used the java.awt.* classes and the jpeg codec package produced by Sun.

4.5 Project Timeline

This section describes our project timeline as we saw it at the beginning of the project cycle. The following timeline was created on February 14th and served as our guide to trying for project completion by May 7th , a week before the due date.

February 14:	Group meeting: complete White Paper
February 18:	White Paper due
March 7:	Begin CILY grammar
March 12:	Finish LRM outline
March 19:	LRM individual parts due; Begin compilation
March 27:	LRM shipped
March 28:	Finish ANTLR-correct grammar / lexer
	Start project plan
	Setup CVS repository
April 3:	Finish Lexer
	Start Parser
	Finish Project Plan
April 11:	Parser code complete, parser bug-fixing
	Start architectural design
April 18:	Ship Parser
	Start Semantic Analyzer
	Finish Architectural design
	Start writing test plan
April 25:	Finish Test Plan
	Start Language Tutorial
	Update LRM and White paper
May 1:	Last day of class; Final exam
May 2:	Finish Language Tutorial
	Do Lessons Learned and Introduction
	Semantic Analyzer code complete
May 5:	Ship the whole thing
	Meet for presentation
May 7:	Finalize final report
May 9:	Finish presentation
May 13:	Give in-class presentation

4.6 Project Log

This portion of the Project Plan outlines what actually happened. In general, the notes that follow in this section are minutes from our weekly meetings. Note that not every meeting had minutes, but one can get a feel as to the progress of our project from the following descriptions.

February 4: Group composition finalized
First contact with Michael Locasto [TA]

February 7: Idea finalized: Image manipulation language
Features: Webpage creator, pixel manipulation
Descriptiors: simple, intuitive, portable, safe, robust

February 13: First meeting with Michael[TA]: discussed the utility and value of such an image manipulation language

February 14: Group meeting
Split up White Paper responsibilities – due February 15th.

Makiko: Motivations for the Language, Features
Jiwan: Background/Related Work, Goals for the Language
Eddie: Intro, Model of Computation
Jon: Proofread and Combine everything

February 18: White Paper due – online submission

February 28: Group Meeting to discuss questions and conerns that we will bring to Michael

March 4: meeting with Michael[TA]

March 7: Group Meeting:
- worked on the preliminary stages of the grammar
- Wrote out an outline for the LRM

March 12: Group Meeting: divided up tasks for LRM
Makiko: Expressions, Declarators, Blocks, Statements
Jiwan: Lexical Convention, Storage Classes, Types, Syntax
Eddie: Functions, Library functions, Lexical scope
Jon: Introduction, Variables, Conversions, Consistency Checking, Compiling & Proofreading

March 13: Group Meeting: grammar work

March 19: LRM parts are due (internal) → move on to compilation

March 23: Sunday after spring break: Group Meeting
Action Items:
- intro: mention automation of image manipulation functions
- appendix: add appendix with grammar
- table of contents: add it
- organize meeting with Michael (email to follow)
- last proofread of LRM

March 27: LRM due – online submission

March 28: Still fixing ANTLR grammar / lexer
Setup CVS repository

April 1: Meeting with Michael[TA] to discuss grammar issues

April 3: Lexer and Parser close to completion
Started Language Tutorial

April 6: Group meeting
Jiwan and Makiko: worked on finalizing lexer and parser
Jon and Eddie: worked on test plan and test cases
Decided to hold off on project plan until next week
Jon works on test cases & Eddie works on automated tests

April 8: Meeting with Michael[TA] to discuss "next step"
Function writing, dividing up modules

April 10: Group meeting: Continued working on lexer and parser
Started Language Tutorial, Test plan

April 11: Code complete Parser: bug fixing
Jiwan, Makiko, Eddie worked on lexer and parser
Jon worked on tutorial more
Planning:
Jiwan and Makiko to continue working on parser
Jon: start project plan, finish tutorial
Eddie: begin writing functions

April 18: Started writing functions

April 25: Finished preliminary test plan
Finishing Language Tutorial
Working on tree parser and code generator

May 1: Last day of class, final exam
Met to discuss final two week timeline

May 2: Almost Completed AST Parser / Code Generator
Architectural Design started

May 7: Finalizing documentation
Group meeting:
- updating & finalizing all documentation
- hope to be done by saturday
- finished all code: only bug fixing left
- still testing

May 8: Meeting to discuss final duties
Going to prepare presentation
Update all documentation
Final testing and bug fixings for the weekend

May 9: Meeting to discuss presentation
Final regression testing and bug fixing

May 13: Presentation!

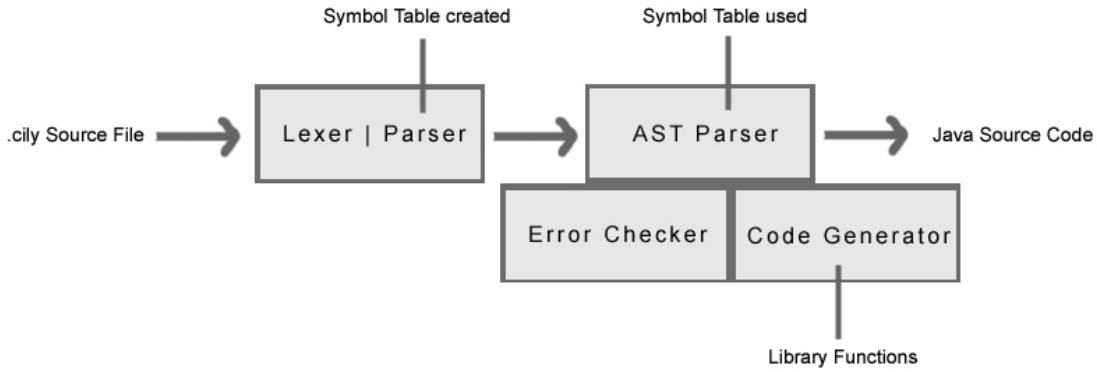
Chapter 5

Architectural Design

5.1 Design of CILY Compiler

5.1.1 Overview

The CILY compiler (cilyc) consists of four major components: CILY Lexer, CILY Parser, AST Parser, and Code Generator. There are also three sub-components, the symbol table, the error checker, and the CILY library, which are used to assist these components. The .cily source file first passes the syntactic analysis of the CILY Lexer and CILY Parser, and the abstract syntax tree (AST) created during the syntactic analysis is semantically checked by the AST Parser. The AST Parser then uses the Code Generator to output the Java source code.



5.1.2 CILY Lexer

The CILY Lexer, implemented using ANTLR, defines many of the fundamental tokens that a .cily source file may contain. Although keywords are not included in the CILY Lexer, other tokens such as operators, numerical constants, string constants, and comments are defined as a sequence of specific characters. Since no other component can detect the newline character, the CILY Lexer keeps a

count of the number of newline characters seen, which is used by the CILY Parser to number the tokens with the appropriate line number.

The CILY Lexer was implemented by Jiwan Choi, Eddie Ishak, and Makiko Yasui.

5.1.3 CILY Parser

The CILY Parser, implemented using ANTLR, defines the rules for the sequence of tokens that a .cily source file may contain. All possible tokens, including all fundamental tokens defined in CILY Lexer, the keywords, and the names of all rules defined in CILY Parser, are numbered in the CILYTokenTypes, which ANTLR automatically creates. Each rule defines the possible sequence of tokens that are valid in the CILY grammar. As the .cily source file is read one token at a time, the parser checks whether the current token and the next token match a certain rule. If the tokens do not match any rule, a syntactic error is observed in the .cily file. For each token that matches a rule, the CILY Parser creates a node in the abstract syntax tree. This AST is used by the AST Parser.

The CILY Parser was implemented by Jiwan Choi, Eddie Ishak, and Makiko Yasui.

5.1.4 AST Parser

The AST Parser, implemented using ANTLR and embedded Java code, parses the AST created by CILY Parser to check the semantics of the .cily source file. With a syntactically incorrect .cily source file, the compiler does not use the AST Parser, therefore no semantic checking is performed for a source file that contains a syntactic error. The AST Parser begins parsing the AST from the top-most node, and matches a rule that describes the actions for that node. Each rule defines a sequence of nodes that may follow the root node, and a set of actions that are performed when the nodes are matched. The embedded Java code specifies these actions. As the rules are matched, the AST Parser keeps a symbol table containing information on the types of variables and functions declared.

The AST Parser was implemented by Eddie Ishak, and Makiko Yasui.

5.1.5 Symbol Table

A symbol table is created using ASTScopes. Each ASTScope contains a pointer to the parent scope, a list of variables and functions defined in its scope, and a list of pointers to the children scopes. The names of the variables and functions are matched with the identifiers seen in the AST Parser, and if the current scope does not contain a matching variable or function, those in the parent scope are searched, until a match is found or the scope is the outermost scope. All of the library function declarations are included in the outermost scope.

The Symbol Table was implemented by Eddie Ishak, and Makiko Yasui.

5.1.6 Error Checker

Although there is no single component that checks for errors, error checks are performed during each of the compilation stages. The CILY Lexer reports an error when the .cily source file contains a syntactic error. The CILY Parser reports an error when there are semantic errors in the .cily source file. The CILY Parser uses the error output method defined in CILY Error, while CILY Lexer uses the error reporting method native to ANTLR. See section 3, Error Handling, for more details.

The Error Checker was implemented by Eddie Ishak, and Makiko Yasui.

5.1.7 Code Generator

The code generator, implemented in Java, outputs the target Java code and an executable file that runs the created target code. The code generation is performed parallel with the AST Parser. As the AST Parser checks the semantic correctness of the .cily source code, it translates the .cily code into Java code and calls the code generator to output a Java code. If the AST Parser observes a semantic error, the generated code will most likely contain an error. The output Java code, if necessary, will call the CILY Library functions to perform the operations specified in the .cily source code.

The Code Generator was implemented by Eddie Ishak, and Makiko Yasui.

5.1.8 CILY Library

The CILY Library consists of static functions that are called by the final output of the code generator. These library functions facilitate code generation, since these methods do not need to be generated for each .cily source file. All image manipulation functions are implemented in this library.

The CILY Library was implemented by Jiwan Choi, Eddie Ishak, Jon Liu, and Makiko Yasui.

5.2 Input and Output

The CILY compiler requires an input of a file with a .cily extension. After syntactic and semantic checking performed by CILY Lexer, CILY Parser, and AST Parser, the code generator will output a Java source code that performs the same operations as the given .cily source file.

5.3 Error Handling

5.3.1 Error Checking

Errors are checked at every compilation stage. During the syntactic analysis phase, syntactic errors are defined as a sequence of tokes that does not match the rules defined in CILY Parser. The ANTLR generated Java code checks for these errors automatically using a try-catch block for each rule. During the semantic analysis phase, errors such as a type mismatch, an array dimension mismatch, a reference to an undeclared variable or a function, a call to a function with the wrong number of arguments, or an attempt to use certain operators with types that are not supported are checked. The embedded Java code in the AST Parser performs these checks. The runtime errors such as an out of bounds array index and file input/output errors are caught in the Java source code using a try-catch block.

5.3.2 Error Response

Both the CILY Parser and AST Parser output error messages when errors are observed. The original reportError method used by ANTLR has been overwritten in the CILY Parser to throw an error and output an error message that specifies the line number containing the error. AST Parser uses the printError method in CILYError to output an error message whenever a semantic error is found. The output error messages contain a brief description of the error as well as the line number containing the error.

5.3.3 Error Recovery

The CILY Parser outputs all syntactic errors found in a given .cily source code. The AST Parser will not perform a semantic check unless the input to the compiler is free of syntactic errors, since a syntactically incorrect code will not generate the correct AST. During the semantic analysis, errors found are output then ignored to check for further errors. Although a single semantic error may cause errors in other parts of the code, the semantic analysis will not stop until the entire AST has been parsed, and all the errors have been found.

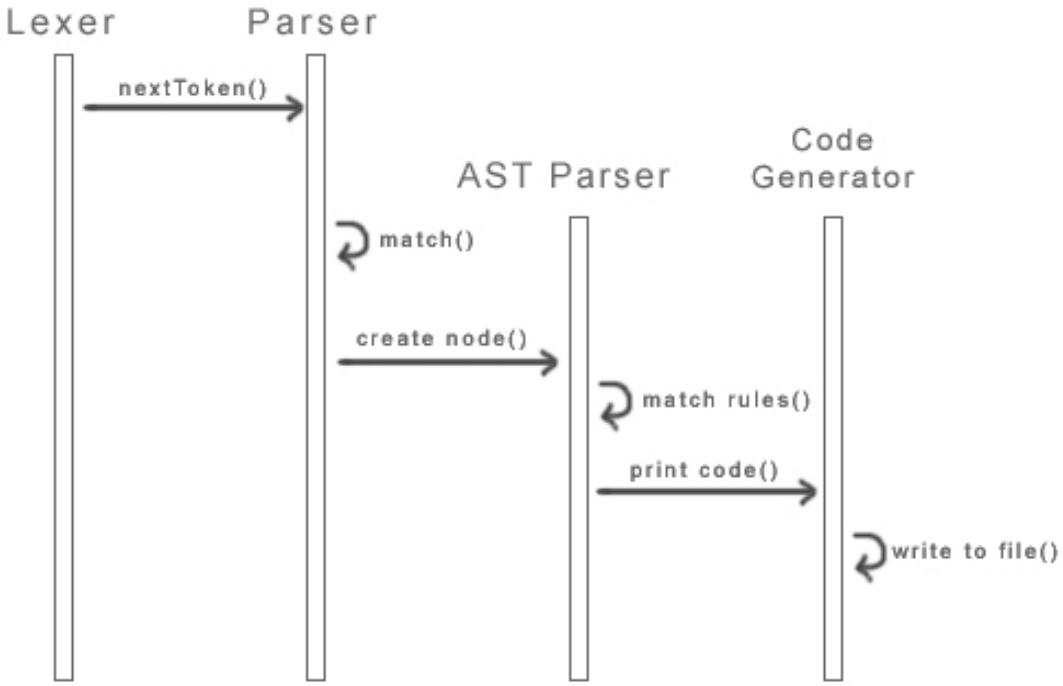
5.4 Interaction Between Components

In parsing the original .cily source code, the CILY Parser uses CILY Lexer to create a list of valid tokens. ANTLR generates a numbered list of tokens from the lexer, the string literals used in the parser, and the rules defined in the parser. The parser refers to these tokens when it uses the match() method to match the Unicode characters in the .cily source code with the tokens. The lexer's nextToken() method returns the next token to be analyzed by the parser.

The interface between the CILY Parser and the AST Parser is the abstract syntax tree that is passed from the CILY Parser to the AST Parser. For every matching rule in the CILY Parser, a root node is generated with information about the contents of the token, including the name, the rule it matched, and the line number relative to the .cily source code. The AST Parser simply parses the tree from the root node, matching nodes to rules and subrules. It uses the name of the token to find the matching rule, and when an error is observed, it uses the line number embedded inside the node name to output an error message with the appropriate line number.

The AST Parser and the code generator work simultaneously during the semantic checking phase. When certain rules are reached, the parser calls the generator with the appropriate print* method matching the rule to output part of the Java source code. The generator opens a file, processes the input from the parser, and writes to the file, until the parser finishes parsing the tree.

The generated Java code may or may not contain references to CILYLibrary.* methods. By having these static methods defined in a single library, the generator can simply output the line, CILYLibrary.* to access all the methods needed for image manipulation. The modularity of the library is useful when modifications are made to the library methods, or new methods are added to the library.



nb: functions are pseudo-code functions except nextToken()

Chapter 6

Test Plan

6.1 Expectations

The designers of CILY had expectations to build a robust and safe system. CILY programs should be in no danger of crashing or failing in compiling or running. The CILY compiler should be able to take any file in CILY syntax and translate it into java target code. The test plan hopes to achieve this by testing every component and piece of code. The test cases involve every aspect of the CILY language in an attempt to provide a universal robustness to any piece of CILY code that is shipped.

6.2 Testing strategy

The CILY developers used a three-stage test plan to sort out the bugs and errors in the translator. The tests were run on the CILY translator while the product was still in parts. We felt the best test strategy would be to test the different modules and components as they were being built instead of opting for the novice Build and Fix model for development.

6.3 Three-Stage Testing

Our testing plan involves a three stage strategy. The first stage involves checking syntax; the targets of analysis are the lexer and parser. In the second stage of testing, the semantic analyzer will go under scrutiny. After both syntax and semantics have been checked for bugs, we will perform a final overall test with more complex CILY programs.

6.3.1 Stage 1 - Syntax: Lexer and Parser

In this first stage of testing, the test cases involve checking the ability of our lexer and parser to correctly evaluate the syntax of a CILY program. At these early stages, the testing style generally used will be white-box testing. Within this

syntax testing, there are two distinct phases. The first phase is the using positive test cases against our translator. These test cases will make sure the compiler can correctly evaluate code that is syntactically correct. The logical second phase is to test the compiler to make sure that it can correctly identify faulty CILY syntax. It is crucial in testing to make sure that the lexer and parser can not only affirm that code has correct syntax, but also recognize code whose syntax is incorrect.

6.3.2 Stage 2 – Semantics: Semantic Analyzer and Symbol Table

Once syntax is verified by the lexer and parser, the tree parser takes another run through the CILY program to ensure that it is semantically correct. This includes checking undeclared/redeclared variables, assigning values to constants, assigning values of invalid types to variables, etc.

6.3.3 Stage 3 – Final Test Phase: More Complex Programs

This final stage of testing will examine how the compiler holds up to more complex CILY programs. In this phase we will examine the run-time environment. All testing for syntax and semantics will be complete and only efficiency and optimization is left to consider.

6.4 Automation

Automation of regression tests was performed on test cases after each major change in the CILY grammar was made. This was to ensure that a change did not break another component of the grammar. The test cases chosen were a breadth representation of all of CILY's language features. Here is the initial list of test cases compiled:

Syntax Test Cases

- declarations
    - arrays, variables, functions
    - with and without assignments
- statements
    - control flow
    - whitespace usage properly handled (space, newline, tab)
    - expressions
        - nested expressions
        - testing all operators (assignment, arithmetic, logical, dot, relational, unary)
    - conditional
    - nested
- function calls

    - with and without parameters
    - nested
    - return types
- global variables
- comments
- string
    - breaking up string on two lines
    - nested quotes
- negative
    - missing semicolons, commas
    - no return type in function declaration
    - mismatching parens, brackets, curly braces
    - using keyword as an id
    - array element access using non int iterator

Semantic Test Cases

- basic test cases
    - changing value of a variable
    - foreach iterate through an array
    - expression using function call returning a value
    - test library functions
    - test operators
    - checking constant and global variable values
- precedence
    - arithmetic precedence
- declarations
    - using undeclared variables, functions
    - redeclaring variables in same scope
- advanced semantic checking
    - writing to constant values
    - implicit casting, disallow explicit casting
    - operator type mismatch
        - arithemetic, logical operators
- function testing
    - no return statement or mismatch return type in function definition
    - argument list errors – mismatch of argument number and types
- complex types
    - non existing fields
    - existing fields
- scope
    - accessing a variable out of scope
    - redeclaring a variable in same scope
    - declaring variable with same name as id in outer scope, different scope

Each test case (something.cily) had an associated expected output file (something.exp), and an actual output by the CILY compiler (something.out). The exp and out file were compared to ensure that the CILY compiler caught expected errors and didn't catch any unexpected errors. Only syntax error checking was automated. Semantic error checking was done manually.

Here is an example output of the test case automation program in verbose mode:

```
Compiling testcases/array/inconsistent_dimensions_01.cily ...  done!
Compiling testcases/array/inconsistent_dimensions_02.cily ...  done!
Compiling testcases/array/array_assignment.cily ...  done!
Compiling testcases/functions/functions_with_no_parameters.cily ...
done!
Compiling testcases/functions/functions_with_parameters.cily ...  done!
Compiling testcases/functions/nested_functions.cily ...  done!
Compiling testcases/functions/return_types.cily ...  done!
Compiling testcases/functions/missing_keyword.cily ...  done!
Compiling testcases/functions/missing_return.cily ...  done!
Compiling testcases/functions/missing_arg_types.cily ...  done!
Compiling testcases/functions/overloading_functions.cily ...  done!
Compiling testcases/functions/returning_array.cily ...  done!
Compiling testcases/functions/array_as_parameter.cily ...  done!
Compiling testcases/functions/constructor.cily ...  done!
Compiling testcases/functions/library_functions.cily ...  done!
Compiling testcases/functions/overloaded_library_functions.cily ...
done!
Compiling testcases/global/basic_global_access.cily ...  done!
Compiling testcases/negative/keyword_as_id.cily ...  done!
Compiling testcases/negative/mismatching_brackets.cily ...  done!
Compiling testcases/negative/mismatching_curly_braces.cily ...  done!
Compiling testcases/negative/mismatching_parens.cily ...  done!
Compiling testcases/negative/missing_comma.cily ...  done!
Compiling testcases/negative/missing_semicolon.cily ...  done!
Compiling testcases/negative/no_return_type.cily ...  done!
Compiling testcases/negative/non_int_array_iterator.cily ...  done!
Compiling testcases/string/break_up_string.cily ...  done!
Compiling testcases/string/nested_quotes.cily ...  done!
Compiling testcases/string/printing_variables.cily ...  done!
Compiling testcases/arrow/arrow_test.cily ...  done!
Compiling testcases/types/primitive_types.cily ...  done!
Compiling testcases/types/mismatching_types.cily ...  done!
Compiling testcases/types/precedence.cily ...  done!
Compiling testcases/types/subtypes.cily ...  done!
Compiling testcases/types/plusplus.cily ...  done!
Compiling testcases/types/plusequals.cily ...  done!
Compiling testcases/loops/foreach_with_images.cily ...  done!
Compiling testcases/loops/while_non_boolean_exp.cily ...  done!
Test file testcases/loops/while_non_boolean_exp.cily FAILED!
Compiling testcases/operators/allops.cily ...  done!
Test cases have errors!
```

Note that while_non_boolean_exp.cily had an error. If there were no errors, it would have printed "Test cases passed!"

6.5 Implementation

Implementation of the testing was done entirely in java. For every new test case to be added, one line was added to the main test case driver program to indicate the location of the new test case. Then the new test case and its expected CILY compiler output file were created.

6.6 Individual Responsibilities

Compiling an initial list of test cases:
Jon Liu and Edward Ishak
Implementing automation of regression tests:
Edward Ishak
Adding test cases mid-implementation:
Makiko Yasui, Edward Ishak, Jiwan Choi, and Jon Liu
Writing Test Plan:
Jon Liu and Edward Ishak

Chapter 7

Lessons Learned

5.1 Jiwan Choi

I learned about how designing a language requires a very different approach to programming. When we tried compiling our first draft of the grammar, we got a lot of non-determinism errors. It took us quite a while to figure out why we were getting non-determinism, and what to do to fix them. Debugging the non-determinism errors was very different compared to the debugging we have done for other programming projects. I learned that specifying a grammar requires a lot of thinking and planning beforehand. I advise the future teams to spend some time in planning before they actually sit in front computers, and that they break the grammar into smaller pieces and add on progressively.

5.2 Edward Ishak

Throughout the CILY design and implementation, I've learned that it is important to work as a team with one major goal. The goal must be clearly defined, and everyone must be motivated toward that one unified goal. Having a project timeline with major and minor milestones, meeting with our TA (a.k.a. "our boss"), and weekly meetings amongst our group members kept us on track to meet that goal. I've also learned that a compiler cannot be written by one person, but rather it's a group effort. Testing and Developing are necessary and should be done by two different groups to remove biases. Automation of testing is EXTREMELY important and can save a lot of time if the initial effort is put forth. I've also learned that pair programming is quite useful. Many bugs were eliminated simply by having two pair of eyes look at the program being written, instead of just one. Finally, I've learned that working in a group with members that are outgoing, genuine, hardworking, and productive are essential to a successful project.

5.3 Makiko Yasui

In working on this project, I learned of the complexity involved in creating a new language, and the difficulty of specifying the grammar. We encountered

numerous non-determinism errors which we did not expect, and in one situation, we were forced to change the grammar from our original. The ambiguities we carried about our language were magnified as we stepped through each of the errors. Clearing these ANTLR errors forced us to rethink our grammar again and again, understanding each rule clearly before moving on to the next step. Instead of analyzing the positive cases where our grammar would work, I learned to critically analyze our grammar from the opposite perspective, trying to think of cases where our grammar would fail. This helped us to disambiguate our grammar and to correct our rules.

5.4 Jonathan Liu

Communication is the key to a successful project. By keeping the lines of communication open, you easily know what is going on, who is doing what, and how things are going in general. It makes it easy to synchronize and make sure your stuff is coherent with everything else. By having lots of meetings, it almost forced us to have mini-deadlines where small modules and components were due. This really helped facilitate our progress. For future students: keep on top of the work, and make sure weeks don't go by where nothing happens. Maintain progress, even if it is very little.

Chapter 8

Appendix

```
// CILY.g
// Authors: Makiko Yasui, Edward Ishak, Jiwan Choi, Jonathon Liu
class CILYLexer extends Lexer;

options
{
    // lookahead 2 characters
    k = 2;
    charVocabulary = '\3'..'\377';
    exportVocab = Cily;
}

{
    public static int lineNumber = 1;
}

protected DOT:     '.';
NOT:           '!';
PLUSOP:        '+';
MINUSOP:       '-';
OPENPAREN:     '(';
CLOSEPAREN:    ')';
SEMICOLON:     ';';
OPENBRACE:     '{';
CLOSEBRACE:    '}';
OPENBRACKET:       '[';
CLOSEBRACKET:      ']';
COMMA:         ',';

EQUALEQUALOP:      "==";
NOTEQUALOP:    "!=";
ASSIGNEQUALOP:     "=";

PLUSEQUALOP:       "+=";
MINUSEQUALOP:      "-=";
MULTEQUALOP:       "*=";
DIVEQUALOP:    "/=";
MODEQUALOP:    "%=";
PLUSPLUSOP:    "++";
MINUSMINUSOP:      "--";
ARROWOP:       "->";

protected
Digit:         '0'..'9'
           ;

NewLine:       ('\n' | "\r\n" | '\r')
           {
               lineNumber++;
               $setType(Token.SKIP);
           }
           ;

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

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

Comments:       "//" ((~'\n'))* '\n'
            {
                lineNumber++;
                $setType(Token.SKIP);
            }
            ;

StringConstant:         '"'! ( ~('"' | '\n' ) | ('"'! '"' ))* '"'!
            ;

RelationalOp:       '>' ('=')?
            | '<' ('=')?
            ;

LogicalOp:      "&&"
            | "||"
            ;

ArithmeticOp:       ('*' | '/' | '%');


Number:         (Digit)+ (DOT (Digit)*)?
            | DOT (Digit)+
            ;

{
    import java.io.*;
}

class CILYParser extends Parser;

options
{
    k=2;
    buildAST = true;
}


{
    boolean debug = false;
    String nodeName = "";
    static boolean foundSyntaxError = false;
    static PrintWriter fileOut = null;

    public void reportError(RecognitionException ex) {
        foundSyntaxError = true;
        if (fileOut != null)
            fileOut.println("Syntax error at line " + CILYLexer.lineNumber + ": " +
ex.getMessage());
        else
            System.err.println("Syntax error at line " + CILYLexer.lineNumber + ": " +
ex.getMessage());
    }
}


constant:       Number {if (debug) System.err.println("\tmatched constant as Number");}
            { nodeName = "number:" + CILYLexer.lineNumber; #constant = #([CONSTANT,
nodeName], constant); }
            | StringConstant
           { nodeName = "stringConstant:" + CILYLexer.lineNumber; #constant = #([CONSTANT,
nodeName], constant); }
            | booleanConstant
            { nodeName = "booleanConstant:" + CILYLexer.lineNumber; #constant =
#([CONSTANT, nodeName], constant); }
            ;
```

```
booleanConstant:   "true"
           { nodeName = "true"; #booleanConstant = #([BOOLEANCONSTANT, nodeName],
booleanConstant);}
           | "false"
           { nodeName = "false"; #booleanConstant = #([BOOLEANCONSTANT, nodeName],
booleanConstant);}
           ;

varType:       "int"!     { nodeName = "int:" + CILYLexer.lineNumber; #varType =
#([VARTYPE, nodeName], varType); }
           | "double"!    { nodeName = "double:" + CILYLexer.lineNumber; #varType =
#([VARTYPE, nodeName], varType); }
           | "string"!{ nodeName = "string:" + CILYLexer.lineNumber; #varType =
#([VARTYPE, nodeName], varType); }
           | "boolean"!   { nodeName = "boolean:" + CILYLexer.lineNumber; #varType =
#([VARTYPE, nodeName], varType); }
           | "void"!  { nodeName = "void:" + CILYLexer.lineNumber; #varType = #([VARTYPE,
nodeName], varType); }
           | "Color"!{ nodeName = "Color:" + CILYLexer.lineNumber; #varType = #([VARTYPE,
nodeName], varType); }
           | "Pixel"!{ nodeName = "Pixel:" + CILYLexer.lineNumber; #varType = #([VARTYPE,
nodeName], varType); }
           | "Image"!{ nodeName = "Image:" + CILYLexer.lineNumber; #varType = #([VARTYPE,
nodeName], varType); }
           | "Rectangle"! { nodeName = "Rectangle:" + CILYLexer.lineNumber; #varType =
#([VARTYPE, nodeName], varType); }
           | "Point"!{ nodeName = "Point:" + CILYLexer.lineNumber; #varType = #([VARTYPE,
nodeName], varType); }
           | "Oval"!  { nodeName = "Oval:" + CILYLexer.lineNumber; #varType = #([VARTYPE,
nodeName], varType); }
           ;

declaration:       varType (OPENBRACKET CLOSEBRACKET! (OPENBRACKET CLOSEBRACKET!)?)? Id
           { nodeName = "declaration:" + CILYLexer.lineNumber; #declaration =
#([DECLARATION, nodeName], declaration); }
           ;

arglist:       declaration (COMMA! declaration)*
           { nodeName = "arglist:" + CILYLexer.lineNumber; #arglist = #([ARGLIST,
nodeName], arglist); }
           | { nodeName = "arglist:" + CILYLexer.lineNumber; #arglist = #([ARGLIST,
nodeName], arglist); } // empty
           ;

expression:    expression1 expression1tail {if (debug) System.err.println("\tmatched
expression");}
          { nodeName = "expression:" + CILYLexer.lineNumber; #expression = #([EXPRESSION,
nodeName], expression); }
           ;

expression1tail:   LogicalOp expression1 expression1tail {if (debug)
System.err.println("\tmatched expression1tail as LogOp expr1");}
           { nodeName = "logicalexpression:" + CILYLexer.lineNumber; #expression1tail =
#([EXPRESSION1TAIL, nodeName], expression1tail); }
           | { nodeName = "logicalexpression_empty:" + CILYLexer.lineNumber;
#expression1tail = #([EXPRESSION1TAIL, nodeName], expression1tail); }
             {if (debug) System.err.println("\tmatched expression1tail as empty");} //
empty
           ;

expression1:       expression2 expression2tail
           { nodeName = "expression1:" + CILYLexer.lineNumber; #expression1 =
#([EXPRESSION1, nodeName], expression1); }
           ;

expression2tail:   (ASSIGNEQUALOP | PLUSEQUALOP | MINUSEQUALOP | MULTEQUALOP | DIVEQUALOP
| MODEQUALOP ) {if (debug) System.err.println("\tmatched assignment op");} expression2
expression2tail
           { nodeName = "assignmentexpression:" + CILYLexer.lineNumber; #expression2tail
= #([EXPRESSION2TAIL, nodeName], expression2tail); }
```

```
            | {if (debug) System.err.println("\tmatched expression3tail as empty");} //
empty
            { nodeName = "assignmentexpression_empty:" + CILYLexer.lineNumber;
#expression2tail = #([EXPRESSION2TAIL, nodeName], expression2tail); }
            ;

expression2:       expression3 expression3tail {if (debug) System.err.println("\tmatched
expression2");}
            { nodeName = "expression2:" + CILYLexer.lineNumber; #expression2 =
#([EXPRESSION2, nodeName], expression2); }
            ;

expression3tail:   (RelationalOp | EQUALEQUALOP | NOTEQUALOP) expression3 expression3tail
{if (debug) System.err.println("\tmatched relationalop AS SOOOOOMTHING");}
            { nodeName = "relationalexpression:" + CILYLexer.lineNumber; #expression3tail
= #([EXPRESSION3TAIL, nodeName], expression3tail); }
            | {if (debug) System.err.println("\tmatched relationalop as EEEEEEEEMPTY");}
// empty
            { nodeName = "relationalexpression_empty:" + CILYLexer.lineNumber;
#expression3tail = #([EXPRESSION3TAIL, nodeName], expression3tail); }
            ;

expression3:       expression4 expression4tail {if (debug) System.err.println("\tmatched
expression3");}
            { nodeName = "expression3:" + CILYLexer.lineNumber; #expression3 =
#([EXPRESSION3, nodeName], expression3); }
            ;

expression4tail:   (PLUSOP | MINUSOP) expression4 expression4tail
            { nodeName = "arithmeticexpression_addsub:" + CILYLexer.lineNumber;
#expression4tail = #([EXPRESSION4TAIL, nodeName], expression4tail); }
            |{ nodeName = "arithmeticexpression_addsub_empty:" + CILYLexer.lineNumber;
#expression4tail = #([EXPRESSION4TAIL, nodeName], expression4tail); } // empty
            ;

expression4:       expression5 expression5tail {if (debug) System.err.println("\tmatched
expression4");}
            { nodeName = "expression4:" + CILYLexer.lineNumber; #expression4 =
#([EXPRESSION4, nodeName], expression4); }
            ;

expression5tail:   ArithmeticOp expression5 expression5tail
            { nodeName = "arithmeticexpression_muldivmod:" + CILYLexer.lineNumber;
#expression5tail = #([EXPRESSION5TAIL, nodeName], expression5tail); }
            |{ nodeName = "arithmeticexpression_muldivmod_empty:" + CILYLexer.lineNumber;
#expression5tail = #([EXPRESSION5TAIL, nodeName], expression5tail); } // empty
            ;

expression5:       { if (debug) System.err.println("in expression5 before matching +/-/!
ravlaue"); } (PLUSOP | MINUSOP | NOT)? {if (debug) System.err.println("in expression5
before matching rvalue");} rvalue {if (debug) System.err.println("\tmatched expression5
as rvalue");}
            { nodeName = "expression5:" + CILYLexer.lineNumber; #expression5 =
#([EXPRESSION5, nodeName], expression5); }
            ;

rvalue:        lvalue {if (debug) System.err.println("\tmatched lvalue");}
            { nodeName = "rvalue_lvalue:" + CILYLexer.lineNumber; #rvalue = #([RVALUE,
nodeName], rvalue); }
            | OPENPAREN! expression CLOSEPAREN! {if (debug) System.err.println("\tmatched
rvalue as parenthesized expr");}
            { nodeName = "rvalue_parens:" + CILYLexer.lineNumber; #rvalue = #([RVALUE,
nodeName], rvalue); }
            | functioncall {if (debug) System.err.println("\tmatched rvalue as
functioncall");}
            { nodeName = "rvalue_functioncall:" + CILYLexer.lineNumber; #rvalue =
#([RVALUE, nodeName], rvalue); }
            | constant {if (debug) System.err.println("\tmatched rvalue as constant");}
            { nodeName = "rvalue_constant:" + CILYLexer.lineNumber; #rvalue = #([RVALUE,
nodeName], rvalue); }
            ;
```

```
lvalue:         Id {if (debug) System.err.println("\tmatched id as lvalue");} lvaluetail
            {
                if (debug) System.err.println("\treached lvalue = ID lvaluetail");
                nodeName = "lvalue:" + CILYLexer.lineNumber;
                #lvalue = #([LVALUE, nodeName], lvalue);
            }
            ;

lvaluetail:     (OPENBRACKET! expression CLOSEBRACKET! (OPENBRACKET! expression
CLOSEBRACKET!)?) lvaluetailtail
            { nodeName = "lvaluetail_arrayaccess:" + CILYLexer.lineNumber; #lvaluetail =
#([LVALUETAIL, nodeName], lvaluetail); }
            | lvaluetailtail
            { nodeName = "lvaluetail_empty:" + CILYLexer.lineNumber; #lvaluetail =
#([LVALUETAIL, nodeName], lvaluetail); }
            ;

lvaluetailtail:         PLUSPLUSOP
            { nodeName = "lvaluetailtail_plusplus:" + CILYLexer.lineNumber;
#lvaluetailtail = #([LVALUETAILTAIL, nodeName], lvaluetailtail); }
            | MINUSMINUSOP
            { nodeName = "lvaluetailtail_minusminus:" + CILYLexer.lineNumber;
#lvaluetailtail = #([LVALUETAILTAIL, nodeName], lvaluetailtail); }
            | (ARROWOP Id lvaluetailtail)
            { nodeName = "lvaluetailtail_arrow:" + CILYLexer.lineNumber; #lvaluetailtail =
#([LVALUETAILTAIL, nodeName], lvaluetailtail); }
            | // empty
            { nodeName = "lvaluetailtail_empty:" + CILYLexer.lineNumber; #lvaluetailtail =
#([LVALUETAILTAIL, nodeName], lvaluetailtail); }
            ;

paramlist:      expression (COMMA! expression)*  { nodeName = "paramlist:" +
CILYLexer.lineNumber; #paramlist = #([PARAMLIST, nodeName], paramlist); }
            | { nodeName = "paramlist_empty:" + CILYLexer.lineNumber; #paramlist =
#([PARAMLIST, nodeName], paramlist); } // empty
            ;

statement:      whilestmt { nodeName = "whilestmt:" + CILYLexer.lineNumber; #statement =
#([STATEMENT, nodeName], statement); }
            | foreachstmt  { nodeName = "foreachstmt:" + CILYLexer.lineNumber; #statement
= #([STATEMENT, nodeName], statement); }
            | ifelsestmt  { nodeName = "ifelsestmt:" + CILYLexer.lineNumber; #statement =
#([STATEMENT, nodeName], statement); }
            | functioncallstmt  { nodeName = "functioncallstmt:" + CILYLexer.lineNumber;
#statement = #([STATEMENT, nodeName], statement); }
            | assignmentstmt  { nodeName = "assignmentstmt:" + CILYLexer.lineNumber;
#statement = #([STATEMENT, nodeName], statement); }
            | printstmt  { nodeName = "printstmt:" + CILYLexer.lineNumber; #statement =
#([STATEMENT, nodeName], statement); }
            ;

printstmt:      "print"! OPENPAREN! expression CLOSEPAREN! SEMICOLON!
            { nodeName = "printstmt:" + CILYLexer.lineNumber; #printstmt = #([PRINTSTMT,
nodeName], printstmt); }
            |"println"! OPENPAREN! expression CLOSEPAREN! SEMICOLON!
            { nodeName = "printlnstmt:" + CILYLexer.lineNumber; #printstmt = #([PRINTSTMT,
nodeName], printstmt); }
            ;

whilestmt:      "while"! OPENPAREN! expression CLOSEPAREN! OPENBRACE! blockofstatements
CLOSEBRACE!
            { nodeName = "whilestmt:" + CILYLexer.lineNumber; #whilestmt = #([WHILESTMT,
nodeName], whilestmt); }
            ;

foreachstmt:        "foreach"! Id "in"! lvalue OPENBRACE! blockofstatements CLOSEBRACE!
            { nodeName = "foreachstmt:" + CILYLexer.lineNumber; #foreachstmt =
#([FOREACHSTMT, nodeName], foreachstmt); }
            ;
```

```
ifelsestmt:    "if"! OPENPAREN! expression CLOSEPAREN! OPENBRACE! blockofstatements
CLOSEBRACE!
           ("elseif"! OPENPAREN! expression CLOSEPAREN! OPENBRACE! blockofstatements
CLOSEBRACE!)*
           ("else"! OPENBRACE! blockofstatements CLOSEBRACE!)?
          { nodeName = "ifelsestmt:" + CILYLexer.lineNumber; #ifelsestmt = #([IFELSESTMT,
nodeName], ifelsestmt); }
           ;

functioncallstmt:  functioncall SEMICOLON!
           { nodeName = "functioncallstmt:" + CILYLexer.lineNumber; #functioncallstmt =
#([FUNCTIONCALLSTMT, nodeName], functioncallstmt); }
           ;

functioncall:      (Id | varType) OPENPAREN! paramlist CLOSEPAREN!
           { nodeName = "functioncall:" + CILYLexer.lineNumber; #functioncall =
#([FUNCTIONCALL, nodeName], functioncall); }
           ;

returnstmt:    "return"! (expression)? SEMICOLON! {if (debug)
System.err.println("\tmatched return statement");}
          { nodeName = "returnstmt:" + CILYLexer.lineNumber; #returnstmt = #([RETURNSTMT,
nodeName], returnstmt); }
           ;

assignmentstmt:       lvalue ((ASSIGNEQUALOP | PLUSEQUALOP | MINUSEQUALOP | MULTEQUALOP |
DIVEQUALOP | MODEQUALOP) {if (debug) System.err.println("\tEQUALOP matched");}
expression)?   SEMICOLON!
           { nodeName = "assignmentstmt:" + CILYLexer.lineNumber; #assignmentstmt =
#([ASSIGNMENTSTMT, nodeName], assignmentstmt); }
           ;

blockofstatements: (arraydeclaration | vardeclaration | statement) blockofstatementstail
           { nodeName = "blockofstatements:" + CILYLexer.lineNumber; #blockofstatements =
#([BLOCKOFSTATEMENTS, nodeName], blockofstatements);}
           | { nodeName = "blockofstatements:" + CILYLexer.lineNumber; #blockofstatements
= #([BLOCKOFSTATEMENTS, nodeName], blockofstatements);} // empty
           ;

blockofstatementstail: (SEMICOLON!)? (arraydeclaration | vardeclaration | statement)
blockofstatementstail { if (debug) System.err.println("\tmatched
blockofstatementstail2"); }
           | { if (debug) System.err.println("\tmatched blockofstatementstail as
empty"); } // empty
           ;

vardeclaration:       "var"! varType Id (ASSIGNEQUALOP! expression)? SEMICOLON!
           { nodeName = "vardeclaration:" + CILYLexer.lineNumber; #vardeclaration =
#([VARDECLARATION, nodeName], vardeclaration); }
           ;

arraylist:     OPENBRACE ((arraylist)* | expression (COMMA expression)*) CLOSEBRACE
           { nodeName = "arraylist:" + CILYLexer.lineNumber; #arraylist = #([ARRAYLIST,
nodeName], arraylist);}
           ;

arraydeclaration:  "array"! varType
           ((OPENBRACKET! expression CLOSEBRACKET! (OPENBRACKET! expression
CLOSEBRACKET!)? Id)
           | (OPENBRACKET CLOSEBRACKET! (OPENBRACKET CLOSEBRACKET!)? Id
           (ASSIGNEQUALOP! (arraylist | expression)))) SEMICOLON!
           { nodeName = "arraydeclaration:" + CILYLexer.lineNumber; #arraydeclaration =
#([ARRAYDECLARATION, nodeName], arraydeclaration);}
           ;

constdeclaration:  "const"! varType Id ASSIGNEQUALOP! constant SEMICOLON!
           { nodeName = "constdeclaration:" + CILYLexer.lineNumber; #constdeclaration =
#([CONSTDECLARATION, nodeName], constdeclaration);}
           ;
```

```
functiondeclaration:   "function"! varType (OPENBRACKET CLOSEBRACKET! (OPENBRACKET
CLOSEBRACKET!)?)? Id OPENPAREN! arglist CLOSEPAREN! OPENBRACE! blockofstatements
returnstmt CLOSEBRACE!
            { nodeName = "functiondeclaration:" + CILYLexer.lineNumber;
#functiondeclaration = #([FUNCTIONDECLARATION, nodeName], functiondeclaration);}
            ;

globaldeclaration: (vardeclaration | constdeclaration | functiondeclaration |
arraydeclaration)+
            { nodeName = "globaldeclaration:" + CILYLexer.lineNumber; #globaldeclaration =
#([GLOBALDECLARATION, nodeName], globaldeclaration);}
            ;

mainfunction:          "main"! OPENPAREN! CLOSEPAREN! OPENBRACE! blockofstatements
CLOSEBRACE!
                    { nodeName = "mainfunction:" + CILYLexer.lineNumber; #mainfunction =
#([MAINFUNCTION, nodeName], mainfunction); }
            ;

program:        (globaldeclaration)?  mainfunction
            { #program = #([PROGRAM], program); }
            ;

{
    import java.util.ArrayList;
}

class CILYTreeParser extends TreeParser;

options
{
    buildAST = true;
}

{
    ASTScope symbolTable = new ASTScope(null, 0);
    ASTScope currentASTScope = symbolTable;
    boolean sawFunction = false;
    int currentReturnType = -1;
    boolean debug = false;
    int numTabs = 0;
    static String filename;
    CodeGenerator codeGen;;
    boolean sawError = false;
}

program:        #(PROGRAM
            {
                codeGen = new CodeGenerator(filename);

                ArrayList varList1 = new ArrayList();
                varList1.add(new ASTVariable("int", "r", null, false));
                varList1.add(new ASTVariable("int", "g", null, false));
                varList1.add(new ASTVariable("int", "b", null, false));
                currentASTScope.add(new ASTFunction("Color", "Color", varList1,
symbolTable, true));

                ArrayList varList2 = new ArrayList();
                varList2.add(new ASTVariable("int", "r", null, false));
                varList2.add(new ASTVariable("int", "g", null, false));
                varList2.add(new ASTVariable("int", "b", null, false));
                varList2.add(new ASTVariable("int", "x", null, false));
                varList2.add(new ASTVariable("int", "y", null, false));
                currentASTScope.add(new ASTFunction("Pixel", "Pixel", varList2,
symbolTable, true));

                ArrayList varList3 = new ArrayList();
                varList3.add(new ASTVariable("int", "x", null, false));
                varList3.add(new ASTVariable("int", "y", null, false));
                currentASTScope.add(new ASTFunction("Point", "Point", varList3,
symbolTable, true));
```

```
                ArrayList varList4 = new ArrayList();
                varList4.add(new ASTVariable("int", "x", null, false));
                varList4.add(new ASTVariable("int", "y", null, false));
                varList4.add(new ASTVariable("int", "width", null, false));
                varList4.add(new ASTVariable("int", "height", null, false));
                currentASTScope.add(new ASTFunction("Rectangle", "Rectangle", varList4,
symbolTable, true));
                currentASTScope.add(new ASTFunction("Oval", "Oval", varList4, symbolTable,
true));
                currentASTScope.add(new ASTFunction("Heart", "Heart", varList4,
symbolTable, true));

                ArrayList varList5 = new ArrayList();
                varList5.add(new ASTVariable("int", "x", null, false));
                varList5.add(new ASTVariable("int", "y", null, false));
                varList5.add(new ASTVariable("int", "base", null, false));
                varList5.add(new ASTVariable("int", "height", null, false));
                currentASTScope.add(new ASTFunction("Triangle", "Triangle", varList5,
symbolTable, true));

                ArrayList varList6 = new ArrayList();
                currentASTScope.add(new ASTFunction("Image", "Image", varList6,
symbolTable, true));

                ArrayList varList7 = new ArrayList();
                varList7.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "Blur", varList7, symbolTable,
true));

                ArrayList varList8 = new ArrayList();
                varList8.add(new ASTVariable("string", "name", null, false));
               currentASTScope.add(new ASTFunction("Image", "Open", varList8, symbolTable,
true));

                ArrayList varList9 = new ArrayList();
                varList9.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "Save", varList9, symbolTable,
true));

                ArrayList varList10 = new ArrayList();
                varList10.add(new ASTVariable("Image", "img", null, false));
                varList10.add(new ASTVariable("string", "name", null, false));
                currentASTScope.add(new ASTFunction("void", "SaveAs", varList10,
symbolTable, true));

                ArrayList varList11 = new ArrayList();
                varList11.add(new ASTVariable("Image", "img", null, false));
                varList11.add(new ASTVariable("double", "level", null, false));
                currentASTScope.add(new ASTFunction("void", "Brighten", varList11,
symbolTable, true));

                ArrayList varList12 = new ArrayList();
                varList12.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "Invert", varList12,
symbolTable, true));

                ArrayList varList13 = new ArrayList();
                varList13.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "Sharpen", varList13,
symbolTable, true));

                ArrayList varList14 = new ArrayList();
                varList14.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "DetectEdges", varList14,
symbolTable, true));

                ArrayList varList15 = new ArrayList();
                varList15.add(new ASTVariable("Image", "img", null, false));
                varList15.add(new ASTVariable("double", "x", null, false));
                varList15.add(new ASTVariable("double", "y", null, false));
```

```
                currentASTScope.add(new ASTFunction("CILYColor", "GetColorAt", varList15,
symbolTable, true));

                ArrayList varList16 = new ArrayList();
                varList16.add(new ASTVariable("Image", "img", null, false));
                varList16.add(new ASTVariable("double", "x", null, false));
                varList16.add(new ASTVariable("double", "y", null, false));
                currentASTScope.add(new ASTFunction("CILYPixel", "GetPixelAt", varList16,
symbolTable, true));

                ArrayList varList17 = new ArrayList();
                varList17.add(new ASTVariable("Image", "dstImage", null, false));
                varList17.add(new ASTVariable("Image", "srcImage", null, false));
               currentASTScope.add(new ASTFunction("void", "Copy", varList17, symbolTable,
true));

                ArrayList varList18 = new ArrayList();
                varList18.add(new ASTVariable("Image", "dstImage", null, false));
                varList18.add(new ASTVariable("Image", "srcImage", null, false));
                varList18.add(new ASTVariable("int", "x", null, false));
                varList18.add(new ASTVariable("int", "y", null, false));
                currentASTScope.add(new ASTFunction("void", "Paste", varList18,
symbolTable, true));

                ArrayList varList19 = new ArrayList();
                varList19.add(new ASTVariable("Image", "img", null, false));
                varList19.add(new ASTVariable("int", "x", null, false));
                varList19.add(new ASTVariable("int", "y", null, false));
                varList19.add(new ASTVariable("int", "w", null, false));
                varList19.add(new ASTVariable("int", "h", null, false));
               currentASTScope.add(new ASTFunction("void", "Crop", varList19, symbolTable,
true));

                ArrayList varList20 = new ArrayList();
                varList20.add(new ASTVariable("Image", "dstImage", null, false));
                varList20.add(new ASTVariable("Rectangle", "rectangle", null, false));
               currentASTScope.add(new ASTFunction("void", "Crop", varList20, symbolTable,
true));

                ArrayList varList21 = new ArrayList();
                varList21.add(new ASTVariable("string", "path", null, false));
                currentASTScope.add(new ASTFunction("Image", "OpenDir", varList21,
symbolTable, true, 1));

                ArrayList varList22 = new ArrayList();
                varList22.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "ConvertToGrayScale",
varList22, symbolTable, true));

                ArrayList varList23 = new ArrayList();
                varList23.add(new ASTVariable("Image", "img", null, false));
                varList23.add(new ASTVariable("Color", "color", null, false));
                varList23.add(new ASTVariable("int", "x", null, false));
                varList23.add(new ASTVariable("int", "y", null, false));
                currentASTScope.add(new ASTFunction("void", "ConvertToGrayScale",
varList23, symbolTable, true));

                ArrayList varList24 = new ArrayList();
                varList24.add(new ASTVariable("Image", "img", null, false));
                varList24.add(new ASTVariable("int", "degrees", null, false));
                currentASTScope.add(new ASTFunction("void", "Rotate", varList24,
symbolTable, true));

                ArrayList varList25 = new ArrayList();
                varList25.add(new ASTVariable("Image", "img", null, false));
                currentASTScope.add(new ASTFunction("void", "FlipHorizontal", varList25,
symbolTable, true));

                ArrayList varList26 = new ArrayList();
                varList26.add(new ASTVariable("Image", "img", null, false));
```

```
                currentASTScope.add(new ASTFunction("void", "FlipVertical", varList26,
symbolTable, true));

                ArrayList varList27 = new ArrayList();
                varList27.add(new ASTVariable("Image", "img", null, false));
                varList27.add(new ASTVariable("Color", "color", null, false));
                varList27.add(new ASTVariable("int", "thickness", null, false));
                currentASTScope.add(new ASTFunction("void", "AddBorder", varList27,
symbolTable, true));

                ArrayList varList28 = new ArrayList();
                varList28.add(new ASTVariable("string", "filename", null, false));
                varList28.add(new ASTVariable("string", "title", null, false));
                varList28.add(new ASTVariable("string", "subtitle", null, false));
                varList28.add(new ASTVariable("Color", "bgcolor", null, false));
                varList28.add(new ASTVariable("Color", "fgcolor", null, false));
                varList28.add(new ASTVariable("int", "style", null, false));
                varList28.add(new ASTVariable("Image", "imgs", null, false, 1));
                currentASTScope.add(new ASTFunction("void", "CreateAlbum", varList28,
symbolTable, true));

                ArrayList varList29 = new ArrayList();
                varList29.add(new ASTVariable("string", "filename", null, false));
                varList29.add(new ASTVariable("string", "title", null, false));
                varList29.add(new ASTVariable("string", "subtitle", null, false));
                varList29.add(new ASTVariable("Color", "bgcolor", null, false));
                varList29.add(new ASTVariable("Color", "fgcolor", null, false));
                varList29.add(new ASTVariable("int", "numPics", null, false));
                varList29.add(new ASTVariable("int", "thumbheight", null, false));
                varList29.add(new ASTVariable("int", "thumbwidth", null, false));
                varList29.add(new ASTVariable("Image", "imgs", null, false, 1));
                currentASTScope.add(new ASTFunction("void", "CreateAlbum", varList29,
symbolTable, true));

                ArrayList varList30 = new ArrayList();
                varList30.add(new ASTVariable("Image", "image", null, false));
                varList30.add(new ASTVariable("string", "dir", null, false));
                varList30.add(new ASTVariable("int", "newHeight", null, false));
                varList30.add(new ASTVariable("int", "newWidth", null, false));
                currentASTScope.add(new ASTFunction("void", "CreateThumbnail", varList30,
symbolTable, true));

                ArrayList varList31 = new ArrayList();
                varList31.add(new ASTVariable("string", "path", null, false));
                currentASTScope.add(new ASTFunction("string", "GetRelPath", varList31,
symbolTable, true));

                ArrayList varList32 = new ArrayList();
                varList32.add(new ASTVariable("string", "path", null, false));
                currentASTScope.add(new ASTFunction("string", "GetDir", varList32,
symbolTable, true));

                ArrayList varList33 = new ArrayList();
                varList33.add(new ASTVariable("string", "dir", null, false));
                varList33.add(new ASTVariable("string", "name", null, false));
                currentASTScope.add(new ASTFunction("string", "GetThumbName", varList33,
symbolTable, true));

                ArrayList varList34 = new ArrayList();
                varList34.add(new ASTVariable("Image", "img", null, false));
                varList34.add(new ASTVariable("Pixel", "pixels", null, false, 1));
                currentASTScope.add(new ASTFunction("Pixel", "SelectInverse", varList34,
symbolTable, true, 1));

                ArrayList varList35 = new ArrayList();
                varList35.add(new ASTVariable("Image", "img", null, false));
                varList35.add(new ASTVariable("Color", "color", null, false));
                varList35.add(new ASTVariable("int", "tolerance", null, false));
                currentASTScope.add(new ASTFunction("Pixel", "MagicWand", varList35,
symbolTable, true, 1));
```

```
                ArrayList varList36 = new ArrayList();
                varList36.add(new ASTVariable("Image", "img", null, false));
                varList36.add(new ASTVariable("Rectangle", "rect", null, false));
                currentASTScope.add(new ASTFunction("Pixel", "SelectRect", varList36,
symbolTable, true, 1));

                ArrayList varList37 = new ArrayList();
                varList37.add(new ASTVariable("Image", "img", null, false));
                varList37.add(new ASTVariable("Oval", "oval", null, false));
                currentASTScope.add(new ASTFunction("Pixel", "SelectOval", varList37,
symbolTable, true, 1));

                ArrayList varList38 = new ArrayList();
                varList38.add(new ASTVariable("Image", "img", null, false));
                varList38.add(new ASTVariable("Pixel", "pixels", null, false, 1));
                currentASTScope.add(new ASTFunction("void", "Paste", varList38,
symbolTable, true));

                ArrayList varList39 = new ArrayList();
                varList39.add(new ASTVariable("Image", "img", null, false, 1));
                currentASTScope.add(new ASTFunction("void", "SaveAll", varList39,
symbolTable, true));

                ArrayList varList40 = new ArrayList();
                varList40.add(new ASTVariable("Image", "img", null, false));
                varList40.add(new ASTVariable("int", "newWidth", null, false));
                varList40.add(new ASTVariable("int", "newHeight", null, false));
                currentASTScope.add(new ASTFunction("void", "Resize", varList40,
symbolTable, true));

                ArrayList varList41 = new ArrayList();
                varList41.add(new ASTVariable("Image", "dstImage", null, false));
                varList41.add(new ASTVariable("Image", "srcImage", null, false));
                varList41.add(new ASTVariable("int", "x", null, false));
                varList41.add(new ASTVariable("int", "y", null, false));
                varList41.add(new ASTVariable("int", "alpha", null, false));
                currentASTScope.add(new ASTFunction("void", "PasteTransparent", varList41,
symbolTable, true));

                ArrayList varList42 = new ArrayList();
                varList42.add(new ASTVariable("Image", "dstImage", null, false));
                varList42.add(new ASTVariable("Pixel", "pixels", null, false, 1));
                varList42.add(new ASTVariable("int", "alpha", null, false));
                currentASTScope.add(new ASTFunction("void", "PasteTransparent", varList42,
symbolTable, true));

                ASTVariable newVar1 = new ASTVariable("int", "CW_90", null, true);
                currentASTScope.add(newVar1);

                ASTVariable newVar2 = new ASTVariable("int", "CW_180", null, true);
                currentASTScope.add(newVar2);

                ASTVariable newVar3 = new ASTVariable("int", "CW_270", null, true);
                currentASTScope.add(newVar3);

                numTabs = codeGen.printClassBeg(numTabs);
            }
            (globaldeclaration)? mainfunc:mainfunction)
            {
                numTabs = codeGen.printClassEnd(numTabs);
                if (mainfunc == null)
                    sawError = CILYError.printError("main function is missing",
CILYLexer.lineNumber);
            }
            ;

mainfunction:       #(MAINFUNCTION
            {
                numTabs = codeGen.printMainBeg(numTabs);
                sawFunction = true;
```

```
                ASTFunction newASTScope = new ASTFunction("void", "main", new ArrayList(),
symbolTable, false);
                currentASTScope.add(newASTScope);
                currentASTScope = newASTScope;
            } blockofstatements)
            {
                numTabs = codeGen.printMainEnd(numTabs);
                if (!sawError)
                    codeGen.generate();

                currentReturnType = ASTType.VOID;
                if (debug) System.err.println("got to mainfunction");

                if (debug) System.err.println(symbolTable);
                currentASTScope = currentASTScope.getParent();
            }
            ;

globaldeclaration: #(GLOBALDECLARATION (vardeclaration[true] | constdeclaration[true] |
arraydeclaration[true] | functiondeclaration)+)
            {
            }
            ;


vardeclaration[boolean isGlobal]: {ASTExpression expr=null;}
            #(vardec:VARDECLARATION type:VARTYPE name:Id (expr = expression)?)
            {
                if (type.getText().equals("void"))
                    sawError = CILYError.printError("a variable cannot be of type void",
Integer.parseInt(vardec.getText().substring(vardec.getText().indexOf(':') + 1)));
                if (!currentASTScope.withinThisScope(name.getText()))
                {
                    if (expr != null)
                    {
                        if (expr.getType() == ASTType.STRING && expr.getEvaluated())
                            numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "\"" + expr.getValue() + "\"", isGlobal, false);
                        else if (expr.getType() ==
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))))
                            numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), expr.getValue() + "", isGlobal, false);
                        else if (expr.getType() == ASTType.INT  && ASTType.DOUBLE ==
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))))
                            numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), expr.getValue() + "", isGlobal, false);
                        else
                            sawError = CILYError.printError("cannot assign " +
expr.getValue() + " to a variable of type " + type.getText().substring(0,
type.getText().indexOf(':')),
Integer.parseInt(vardec.getText().substring(vardec.getText().indexOf(':') + 1)));
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), expr.getValue(), false));
                    }
                    else
                    {
                        int typeValue = ASTType.getTypeValue(type.getText().substring(0,
type.getText().indexOf(':')));
                        if (typeValue == ASTType.INT)
                        {
                            Integer initVal = new Integer(0);
                            numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "0", isGlobal, false);
                            currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                        }
                        else if (typeValue == ASTType.DOUBLE)
```

```
                    {
                        Double initVal = new Double(0.0);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "0.0", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.BOOLEAN)
                    {
                        Boolean initVal = new Boolean(false);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "false", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.STRING)
                    {
                        String initVal = "";
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "\"\"", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.IMAGE)
                    {
                        CILYImage initVal = new CILYImage();
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "new CILYImage()", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.COLOR)
                    {
                        CILYColor initVal = new CILYColor(0, 0, 0);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), "new CILYColor(0, 0, 0)", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.POINT)
                    {
                        CILYPoint initVal = new CILYPoint(0, 0);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(),  "new CILYPoint(0, 0)", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.PIXEL)
                    {
                        CILYPixel initVal = new CILYPixel(0, 0, 0, 0, 0);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(),  "new CILYPixel(0, 0, 0, 0, 0)", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                    else if (typeValue == ASTType.RECTANGLE)
                    {
                        CILYRectangle initVal = new CILYRectangle(0, 0, 1, 1);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(),  "new CILYRectangle(0, 0, 0, 0)", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
```

```
                    else if (typeValue == ASTType.OVAL)
                    {
                        CILYOval initVal = new CILYOval(0, 0, 0, 0);
                        numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(),  "new CILYOval(0, 0, 0, 0)", isGlobal, false);
                        currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), initVal, false));
                    }
                }
            }
            else
                sawError = CILYError.printError("variable redefinition for " +
name.getText(), Integer.parseInt(vardec.getText().substring(vardec.getText().indexOf(':')
+ 1)));
            if (debug) System.err.println("got to vardecl, type = " +
type.getText().substring(0, type.getText().indexOf(':')) + ", name = " + name.getText());
        }
        ;

constdeclaration[boolean isGlobal]: {ASTExpression cnst=null;}
        #(constdec:CONSTDECLARATION type:VARTYPE name:Id cnst = constant)
        {
            if (!currentASTScope.withinThisScope(name.getText()))
            {
                numTabs = codeGen.printVarDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name.getText(), cnst.getValue() + "", isGlobal, true);
                currentASTScope.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), cnst.getValue(), true));
            }
            else
                sawError = CILYError.printError("variable redefinition for " +
name.getText(),
Integer.parseInt(constdec.getText().substring(constdec.getText().indexOf(':') + 1)));
            if (debug) System.err.println("got to constdecl, type = " +
type.getText().substring(0, type.getText().indexOf(':')));
        }
        ;

arraydeclaration[boolean isGlobal]: {ASTExpression size1=null, size2=null; int
numDimensions = 1; ASTArrayInfo arrayInfo = new ASTArrayInfo(); ASTExpression expr =
null;}
        #(arraydec:ARRAYDECLARATION type:VARTYPE ((size1 = expression (size2 =
expression)? name1:Id) | OPENBRACKET (opbracket:OPENBRACKET)? name2:Id
        {
            if (opbracket != null)
                numDimensions = 2;
        }
        ((arrayInfo = arraylist[ASTType.getTypeValue(type.getText().substring(0,
type.getText().indexOf(':'))), numDimensions, arrayInfo]
        | expr = expression))))
        {
            if (name1 != null) //size specified
            {
                if (!currentASTScope.withinThisScope(name1.getText()))
                {
                    int typeValue = ASTType.getTypeValue(type.getText().substring(0,
type.getText().indexOf(':')));
                    if (size2 == null) //one dimension
                    {
                        if (size1.getType() == ASTType.INT)
                        {
                            if (typeValue == ASTType.INT)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "0", 1, isGlobal);
                            else if (typeValue == ASTType.DOUBLE)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "0.0", 1, isGlobal);
                            else if (typeValue == ASTType.BOOLEAN)
```

```
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "false", 1, isGlobal);
                            else if (typeValue == ASTType.STRING)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "", 1, isGlobal);
                            else if (typeValue == ASTType.IMAGE)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "new CILYImage()", 1, isGlobal);
                            else if (typeValue == ASTType.COLOR)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "new CILYColor(0, 0, 0)", 1,
isGlobal);
                            else if (typeValue == ASTType.POINT)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "new CILYPoint(0, 0)", 1,
isGlobal);
                            else if (typeValue == ASTType.PIXEL)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "new CILYPixel(0, 0, 0, 0, 0)",
1, isGlobal);
                            else if (typeValue == ASTType.RECTANGLE)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "new CILYRectangle(0, 0, 1, 1)",
1, isGlobal);
                            else if (typeValue == ASTType.OVAL)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", null, "new CILYOval(0, 0, 0, 0)", 1,
isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name1.getText(),
false, 1));
                        }
                        else
                        {
                            sawError = CILYError.printError("array size must be an
integer value", CILYLexer.lineNumber);
                            numTabs = codeGen.printArrayDeclaration(numTabs, typeValue,
name1.getText(), size1.getValue() + "", null, 1, null, isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name1.getText(),
false, 1));
                        }
                    }
                    else        //two dimensions
                    {
                        if (size1.getType() == ASTType.INT && size2.getType() ==
ASTType.INT)
                        {
                            if (typeValue == ASTType.INT)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "0", 2,
isGlobal);
                            else if (typeValue == ASTType.DOUBLE)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "0.0", 2,
isGlobal);
                            else if (typeValue == ASTType.BOOLEAN)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "false", 2,
isGlobal);
                            else if (typeValue == ASTType.STRING)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "", 2,
isGlobal);
                            else if (typeValue == ASTType.IMAGE)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "new
CILYImage()", 2, isGlobal);
                            else if (typeValue == ASTType.COLOR)
```

```
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "new
CILYColor(0, 0, 0)", 2, isGlobal);
                            else if (typeValue == ASTType.POINT)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "new
CILYPoint(0, 0)", 2, isGlobal);
                            else if (typeValue == ASTType.PIXEL)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "new
CILYPixel(0, 0, 0, 0, 0)", 2, isGlobal);
                            else if (typeValue == ASTType.RECTANGLE)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "new
CILYRectangle(0, 0, 1, 1)", 2, isGlobal);
                            else if (typeValue == ASTType.OVAL)
                                numTabs = codeGen.printArrayDeclaration(numTabs,
typeValue, name1.getText(), size1.getValue() + "", size2.getValue() + "", "new CILYOval(0,
0, 0, 0)", 2, isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name1.getText(),
false, 2));
                        }
                        else
                        {
                            sawError = CILYError.printError("array size must be an
integer value", CILYLexer.lineNumber);
                            numTabs = codeGen.printArrayDeclaration(numTabs, typeValue,
name1.getText(), size1.getValue() + "", size2.getValue() + "", 2, null, isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name1.getText(),
false, 2));
                        }
                    }
                }
                else
                    sawError = CILYError.printError("variable redefinition for " +
name1.getText(),
Integer.parseInt(arraydec.getText().substring(arraydec.getText().indexOf(':') + 1)));
            }
            else //size not specified
            {
                if (!currentASTScope.withinThisScope(name2.getText()))
                {
                    if (opbracket == null) //one dimension
                    {
                        if (arrayInfo.getValue().equals("") && expr.getArrayDim() == 1
&& expr.getType() == ASTType.getTypeValue(type.getText().substring(0,
type.getText().indexOf(':')))) //expression specified
                        {
                            numTabs = codeGen.printArrayDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name2.getText(), null, null, 1, expr.getValue() + "", isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name2.getText(),
false, 1));
                        }
                        else if (!arrayInfo.getValue().equals(""))
                        {
                            numTabs = codeGen.printArrayDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name2.getText(), null, null, 1, arrayInfo.getValue(), isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name2.getText(),
false, 1));
                        }
                        else
                        {
                            sawError = CILYError.printError(expr.getValue() + " does
not evaluate to a one-dimensional array of type " +
```

```
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
Integer.parseInt(arraydec.getText().substring(arraydec.getText().indexOf(':') + 1)));
                        }
                    }
                    else       //two dimensions
                    {
                        if (arrayInfo.getValue().equals("") && expr.getArrayDim() == 1
&& expr.getType() == ASTType.getTypeValue(type.getText().substring(0,
type.getText().indexOf(':')))) //expression specified
                        {
                            numTabs = codeGen.printArrayDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name2.getText(), null, null, 2, expr.getValue() + "", isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name2.getText(),
false, 2));
                        }
                        else if (!arrayInfo.getValue().equals(""))
                        {
                            numTabs = codeGen.printArrayDeclaration(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
name2.getText(), null, null, 2, arrayInfo.getValue(), isGlobal);
                            currentASTScope.add(new
ASTMyArray(type.getText().substring(0, type.getText().indexOf(':')), name2.getText(),
false, 2));
                        }
                        else
                        {
                            sawError = CILYError.printError(expr.getValue() + " does
not evaluate to a two-dimensional array of type " +
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))),
Integer.parseInt(arraydec.getText().substring(arraydec.getText().indexOf(':') + 1)));
                        }
                    }
                }
                else
                    sawError = CILYError.printError("variable redefinition for " +
name2.getText(),
Integer.parseInt(arraydec.getText().substring(arraydec.getText().indexOf(':') + 1)));
            }
            if (debug) System.err.println("got to arraydecl, type = " +
type.getText().substring(0, type.getText().indexOf(':')));
        }
        ;

arraylist[int type, int numDimensions, ASTArrayInfo arrayInfo] returns [ASTArrayInfo
newArrayInfo = new ASTArrayInfo();]: {ASTExpression expr1=null, expr2=null;}
        #(list:ARRAYLIST OPENBRACE
        {
            newArrayInfo.addValue("{");
        }
        ((
        {
            if (!arrayInfo.getValue().equals(""))
                newArrayInfo.addValue(",");
        }
        arrayInfo = arraylist[type, numDimensions - 1, arrayInfo]
        {
            newArrayInfo.incrementSize2();  //counts number of arraylists
            newArrayInfo.addValue(arrayInfo.getValue());
        }
        )*
        {
            newArrayInfo.setSize1(arrayInfo.getSize1());
        }
        | expr1 = expression
        {
            newArrayInfo.incrementSize1();
            newArrayInfo.addValue(expr1.getValue() + "");
            if (numDimensions != 1)
```

```
                    sawError = CILYError.printError("the initial array value does not match
the array dimension.",
Integer.parseInt(list.getText().substring(list.getText().indexOf(':') + 1)));
                if (type != expr1.getType())
                    sawError = CILYError.printError("array values must be of type " +
ASTType.getStringValue(type),
Integer.parseInt(list.getText().substring(list.getText().indexOf(':') + 1)));
            }
            (COMMA
            {
                newArrayInfo.addValue(",");
            }
            expr2 = expression
            {
                newArrayInfo.incrementSize1();
                newArrayInfo.addValue(expr2.getValue() + "");
                if (type != expr2.getType())
                    sawError = CILYError.printError("array values must be of type " +
ASTType.getStringValue(type),
Integer.parseInt(list.getText().substring(list.getText().indexOf(':') + 1)));
            }
            )*
            {
                if (arrayInfo.getSize1() != 0)
                {
                    if (arrayInfo.getSize1() != newArrayInfo.getSize1())
                        sawError = CILYError.printError("array size is not consistent",
Integer.parseInt(list.getText().substring(list.getText().indexOf(':') + 1)));
                    newArrayInfo.setSize1(arrayInfo.getSize1());
                }
            }
            ) CLOSEBRACE
            {
                newArrayInfo.addValue("}");
            }
            )
            {
                if (debug) System.err.println("got to arraylist");
            }
            ;

functiondeclaration:   #(funcdec:FUNCTIONDECLARATION type:VARTYPE (bracket1:OPENBRACKET
(bracket2:OPENBRACKET)?)? name:Id {ArrayList variables = new ArrayList();} variables =
arglist
            {
                if (debug) System.err.println("got to functiondecl, type = " +
type.getText().substring(0, type.getText().indexOf(':')));
                sawFunction = true;
                currentReturnType = ASTType.getTypeValue(type.getText().substring(0,
type.getText().indexOf(':')));

                if (!currentASTScope.withinThisScope(name.getText()))
                {
                    ASTFunction newASTScope = null;
                    if (bracket1 != null && bracket2 != null)
                    {
                        numTabs = codeGen.printFunctionBeg(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))), 2,
name.getText(), variables);
                        newASTScope = new ASTFunction(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), variables, symbolTable, false, 2);
                    }
                    else if (bracket1 != null)
                    {
                        numTabs = codeGen.printFunctionBeg(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))), 1,
name.getText(), variables);
                        newASTScope = new ASTFunction(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), variables, symbolTable, false, 1);
                    }
                    else
```

```
                    {
                        numTabs = codeGen.printFunctionBeg(numTabs,
ASTType.getTypeValue(type.getText().substring(0, type.getText().indexOf(':'))), 0,
name.getText(), variables);
                        newASTScope = new ASTFunction(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), variables, symbolTable, false);
                    }
                    currentASTScope.add(newASTScope);
                    currentASTScope = newASTScope;
                    for (int i = 0; i < variables.size(); i++)
                    {
                        if (((ASTVariable)variables.get(i)).getArrayDim() > 0)
                            currentASTScope.add(new
ASTMyArray(ASTType.getStringValue(((ASTVariable)variables.get(i)).getType()),
((ASTVariable)variables.get(i)).getName(), ((ASTVariable)variables.get(i)).isConst(),
((ASTVariable)variables.get(i)).getArrayDim()));
                        else
                            currentASTScope.add((ASTVariable)variables.get(i));
                    }
                }
                else
                {
                    ASTFunction newASTScope = new ASTFunction("void", name.getText(), new
ArrayList(), symbolTable, false);
                    currentASTScope.add(newASTScope);
                    currentASTScope = newASTScope;
                    sawError = CILYError.printError("function redefinition for " +
name.getText(),
Integer.parseInt(funcdec.getText().substring(funcdec.getText().indexOf(':') + 1)));
                }
            }
            blockofstatements returnstmt)
            {
                currentReturnType = ASTType.VOID;
                numTabs = codeGen.printFunctionEnd(numTabs);

                if (debug) System.err.println(symbolTable);
                currentASTScope = currentASTScope.getParent();
            }
            ;

arglist returns [ArrayList newVariables=new ArrayList()]:
            #(ARGLIST (newVariables = declaration[newVariables])*)
            {
                if (debug) System.err.println("got to arglist");
            }
            ;

declaration[ArrayList variables] returns [ArrayList newVariables=variables]:
            #(DECLARATION type:VARTYPE (bracket1:OPENBRACKET (bracket2:OPENBRACKET)?)?
name:Id)
            {
                if (debug) System.err.println("got to declaration, type = " +
type.getText().substring(0, type.getText().indexOf(':')) + ", name = " + name.getText());
                if (bracket1 != null && bracket2 != null)
                    newVariables.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), null, false, 2));
                else if (bracket1 != null)
                    newVariables.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), null, false, 1));
                else
                    newVariables.add(new ASTVariable(type.getText().substring(0,
type.getText().indexOf(':')), name.getText(), null, false));
            }
            ;

blockofstatements: #(BLOCKOFSTATEMENTS
            {
                if (debug) System.err.println("got to block of statements");
                if (!sawFunction)
                {
```

```
                    ASTScope newScope = new ASTScope(currentASTScope,
(currentASTScope.getLevel() + 1));
                    currentASTScope.add(newScope);
                    currentASTScope = newScope;
                }
                else
                    sawFunction = false;

            }
            (arraydeclaration[false] | vardeclaration[false] | statement)*)
            ;

statement:     #(STATEMENT {if (debug) System.err.println("got to statement");}
(stmt1:whilestmt | stmt2:foreachstmt | stmt3:ifelsestmt | functioncallstmt |
assignmentstmt | printstmt))
            ;

whilestmt: {ASTExpression expr=null;}
            #(stmt:WHILESTMT
            {
                if (debug) System.err.println("got to while statement");
            }
            expr = expression
            {
                if (expr.getType() != ASTType.BOOLEAN)
                    sawError = CILYError.printError(expr.getValue() + " does not evaluate
to a boolean expression",
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                numTabs = codeGen.printWhileBeg(numTabs, expr.getValue() + "");
            }
            blockofstatements
            {
                numTabs = codeGen.printWhileEnd(numTabs);
                if (debug) System.err.println(symbolTable);
                currentASTScope = currentASTScope.getParent();
            }
            )
            ;

foreachstmt: {ASTExpression lval=null; boolean ok = true;}
            #(stmt:FOREACHSTMT name:Id
            {
                Object obj = currentASTScope.withinScope(name.getText());
                if (obj != null)
                {
                    sawError = CILYError.printError(name.getText() + " has already been
declared", Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                }
                else
                {
                    currentASTScope.add(new ASTVariable("int", name.getText(), new
Integer(0), false));
                }
            }
            lval = lvalue
            {
                if (ok && lval != null)
                {
                    obj = currentASTScope.withinScope(lval.getValue() + "");
                    if (lval.getArrayDim() < 1)
                        ok = false;
                }

                if (ok)
                    numTabs = codeGen.printForeachBeg(numTabs, name.getText(),
lval.getValue() + "");

            }
            blockofstatements)
            {
                numTabs = codeGen.printForeachEnd(numTabs);
```

```
                if (debug) System.err.println(symbolTable);
                currentASTScope = currentASTScope.getParent();

                if (debug) System.err.println("got to foreach statement");
            }
            ;

ifelsestmt: {ASTExpression expr=null; int count = 0;}
            #(stmt:IFELSESTMT (expr = expression
            {
                if (expr.getType() == ASTType.BOOLEAN)
                {
                    if (count == 0)
                        numTabs = codeGen.printIfBeg(numTabs, expr.getValue() + "");
                    else
                        numTabs = codeGen.printElseIf(numTabs, expr.getValue() + "");
                }
                else
                    sawError = CILYError.printError("the expression does not evaluate to a
boolean value", Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') +
1)));
                count++;
            }
            blockofstatements
            {
                numTabs = codeGen.printIfEnd(numTabs);
                if (debug) System.err.println(symbolTable);
                currentASTScope = currentASTScope.getParent();
            }
            )+ (
            {
                numTabs = codeGen.printElse(numTabs);
            }
            elseBlock:blockofstatements
            {
                numTabs = codeGen.printIfEnd(numTabs);

                if (debug) System.err.println(symbolTable);
                currentASTScope = currentASTScope.getParent();
            }
            )?)
            {

                if (debug) System.err.println("got to if else statement");
                if (elseBlock != null)
                    if (debug) System.err.println("got to else case");
            }
            ;

assignmentstmt:    {ASTExpression lval=null, expr=null, retValue=null;}
            #(stmt:ASSIGNMENTSTMT lval = lvalue ((op1:ASSIGNEQUALOP | op2:PLUSEQUALOP |
op3:MINUSEQUALOP | op4:MULTEQUALOP | op5:DIVEQUALOP | op6:MODEQUALOP) expr =
expression)?)
            {
                retValue = lval;
                int arrayDim1 = lval.getArrayDim();
                int arrayDim2 = 0;

                if (expr != null)
                    arrayDim2 = expr.getArrayDim();

                if (lval.isConst())
                   sawError = CILYError.printError(lval.getValue() + " is a const variable,
and its value cannot be changed",
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                else if (!lval.isLvalue() && !lval.isUnaryExpression())
                {
                    sawError = CILYError.printError("the left hand side of this expression
cannot be assigned a value",
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                }
```

```
                else
                {
                    if (op1 != null && arrayDim1 == arrayDim2)
                    {
                        expr.setOp(ASTOp.EQUALS);
                        retValue = lval.evaluate(expr,
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op2 != null && arrayDim1 == arrayDim2)
                    {
                        expr.setOp(ASTOp.PLUSEQUALS);
                        retValue = lval.evaluate(expr,
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op3 != null && arrayDim1 == arrayDim2)
                    {
                        expr.setOp(ASTOp.MINUSEQUALS);
                        retValue = lval.evaluate(expr,
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op4 != null && arrayDim1 == arrayDim2)
                    {
                        expr.setOp(ASTOp.MULTIPLYEQUALS);
                        retValue = lval.evaluate(expr,
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op5 != null && arrayDim1 == arrayDim2)
                    {
                        expr.setOp(ASTOp.DIVIDEEQUALS);
                        retValue = lval.evaluate(expr,
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op6 != null && arrayDim1 == arrayDim2)
                    {
                        expr.setOp(ASTOp.MODEQUALS);
                        retValue = lval.evaluate(expr,
Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (!lval.isUnaryExpression())
                    {
                        sawError = CILYError.printError(lval.getValue() + " is not a
statement", Integer.parseInt(stmt.getText().substring(stmt.getText().indexOf(':') + 1)));
                    }
                    else
                    {
                        retValue = lval;
                    }
                    numTabs = codeGen.printAssignStmt(numTabs, retValue.getValue() + "");
                }
                if (debug) System.err.println("got to assignmentstmt, lvalue = " +
lval.getValue());
            }
            ;

functioncallstmt: {ASTExpression funcCall=null;}
            #(FUNCTIONCALLSTMT funcCall = functioncall)
            {
                numTabs = codeGen.printFunctionCallStmt(numTabs, funcCall.getValue() +
"");
```

```
            if (debug) System.err.println("Got to functioncallstmt");
        }
        ;

functioncall returns [ASTExpression retValue=null]: {ASTFunction[] func=null; ArrayList
objects = new ArrayList();}
        #(funcall:FUNCTIONCALL (name1:Id | type:VARTYPE)
        {
            String name2 = "";

            if (type != null)
                 name2 = type.getText().substring(0, type.getText().indexOf(':'));

            if (name1 != null)
                objects = currentASTScope.withinScopeFunc(new String(name1.getText()),
objects);
            else if (name2 != null && (!name2.equals("int") && !name2.equals("void")
&& !name2.equals("double") && !name2.equals("string") && !name2.equals("boolean")))
                objects = currentASTScope.withinScopeFunc(name2, objects);

            if (objects.size() != 0)
            {
                func = new ASTFunction[objects.size()];
                for (int i = 0; i < objects.size(); i++)
                {
                    func[i] = (ASTFunction)(objects.get(i));
                    if (!name2.equals(""))
                        func[i].setName("new CILY" + name2);
                    else if (func[i].isLibraryFunction())
                        func[i].setName("CILYLibrary." + func[i].getName());
                }
            }
            else
            {
                func = new ASTFunction[1];
                if (name1 != null)
                {
                    func[0] = new ASTFunction("void", name1.getText(), new ArrayList(),
symbolTable, false);
                    sawError = CILYError.printError(name1.getText() + " is not a valid
function", Integer.parseInt(funcall.getText().substring(funcall.getText().indexOf(':') +
1)));
                }
                else
                {
                    func[0] = new ASTFunction("void", name2, new ArrayList(),
symbolTable, false);
                    sawError = CILYError.printError(name2 + " is not a valid function",
Integer.parseInt(funcall.getText().substring(funcall.getText().indexOf(':') + 1)));
                }
            }
        }
        retValue = paramlist[func]
        {
            retValue.setEvaluated(false);
            if (debug) System.err.println("\t got to functioncall rule params = " +
retValue.getValue());
        }
        )
        ;

paramlist[ASTFunction[] func] returns [ASTExpression retValue=new
ASTExpression(func[0].getReturnType(), func[0].getName() + "(", false);]: {ASTExpression
expr=null; int count=0; boolean[] match;}
        #(param:PARAMLIST
        {
            match = new boolean[func.length];
            for (int i = 0; i < match.length; i++)
            {
                match[i] = true;
            }
```

```
            }
            (expr = expression
            {
                boolean hasWritten = false;
                for (int i = 0; i < func.length; i++)
                {
                    if (match[i] && (func[i].getArg(count) == null ||
func[i].getArg(count).getType() != expr.getType() || func[i].getArgDim(count) !=
expr.getArrayDim()))
                        match[i] = false;
                    else if (!hasWritten && match[i])
                    {
                        hasWritten = true;
                        if (count == 0)
                        {
                            if (expr.getType() == ASTType.STRING && expr.getEvaluated())
                                retValue.setValue(retValue.getValue() + "\"" +
expr.getValue() + "\"");
                            else
                                retValue.setValue(retValue.getValue() + "" +
expr.getValue());
                        }
                        else
                        {
                            if (expr.getType() == ASTType.STRING && expr.getEvaluated())
                                retValue.setValue(retValue.getValue() + ", \"" +
expr.getValue() + "\"");
                            else
                                retValue.setValue(retValue.getValue() + ", " +
expr.getValue());
                        }
                    }
                }
                count++;
            }
            )*
            {
                int numMatched = 0;
                int num = 0;
                for (int i = 0; i < match.length; i++)
                {
                    if (match[i])
                    {
                        if (func[i].getArgSize() == count)
                        {
                            numMatched++;
                            num = i;
                        }
                    }
                }
                if (numMatched == 1)
                {
                    retValue.setType(func[num].getReturnType());
                    retValue.setArrayDim(func[num].getArrayDim());
                }
                else
                    sawError = CILYError.printError("the argument[s] for " +
func[0].getName().substring(func[0].getName().indexOf('.') + 1) + " does not match the
function declaration",
Integer.parseInt(param.getText().substring(param.getText().indexOf(':') + 1)));
                retValue.setValue(retValue.getValue() + ")");
            }
            )
            ;


returnstmt:    {ASTExpression expr = null;}
            #(retstmt:RETURNSTMT (expr = expression)? )
            {
                if (expr == null && currentReturnType != ASTType.VOID)
```

```
                    sawError = CILYError.printError("this function must return a value of
type " + ASTType.getStringValue(currentReturnType),
Integer.parseInt(retstmt.getText().substring(retstmt.getText().indexOf(':') + 1)));
                else if (expr != null && expr.getType() == currentReturnType)
                {
                    numTabs = codeGen.printReturnStmt(numTabs, expr.getValue() + "");
                    if (debug) System.err.println("Found a return statment, returning " +
expr.getValue());
                }
                else if (expr == null && currentReturnType == ASTType.VOID)
                {
                    numTabs = codeGen.printReturnStmt(numTabs, "");
                    if (debug) System.err.println("Found a return statment, returning
void");
                }
                else
                    sawError = CILYError.printError("this function must return a value of
type " + ASTType.getStringValue(currentReturnType) + ", not of type " +
ASTType.getStringValue(expr.getType()),
Integer.parseInt(retstmt.getText().substring(retstmt.getText().indexOf(':') + 1)));
            }
            ;

printstmt: {ASTExpression expr=null;}
            #(prnt:PRINTSTMT expr = expression)
            {
                String str = prnt.getText().substring(0, prnt.getText().indexOf(':'));
                if (str.equals("printstmt"))
                {
                    if (expr.getType() == ASTType.STRING && expr.getEvaluated())
                        numTabs = codeGen.printPrintStmt(numTabs, "\"" + expr.getValue() +
"\"");
                    else
                        numTabs = codeGen.printPrintStmt(numTabs, expr.getValue() + "");
                }
                else
                {
                    if (expr.getType() == ASTType.STRING && expr.getEvaluated())
                        numTabs = codeGen.printPrintlnStmt(numTabs, "\"" + expr.getValue()
+ "\"");
                    else
                        numTabs = codeGen.printPrintlnStmt(numTabs, expr.getValue() + "");
                }
            }
            ;

expression returns [ASTExpression retValue=null]: {ASTExpression e1 = null; ASTExpression
e1tail = null;}
            #(EXPRESSION e1 = expression1 e1tail = expression1tail[e1])
            {
                retValue = e1tail;
                if (debug) System.err.println("got to expression");
            }
            ;

expression1tail[ASTExpression exprIn] returns [ASTExpression retValue=exprIn]:
{ASTExpression e1tail=null;  ASTExpression e1 = null;}
            #(tail:EXPRESSION1TAIL (op:LogicalOp e1 = expression1
            {
                if (op != null && exprIn.getArrayDim() == 0)
                {
                    retValue.setLvalue(false);
                    if (op.getText().equals("&&"))
                        e1.setOp(ASTOp.AND);
                    else if (op.getText().equals("||"))
                        e1.setOp(ASTOp.OR);
                    retValue = exprIn.evaluate(e1,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                    if (debug) System.err.println("got to expression1tail - logical");
                }
                else if (op != null && exprIn.getArrayDim() != 0)
```

```
                    sawError = CILYError.printError("Cannot use " + op.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            }
            e1tail = expression1tail[retValue])?)
            {
                if (e1tail != null)
                    retValue = e1tail;
                if (debug) System.err.println("retvalue is " + retValue.getValue());
            }

            ;

expression1 returns [ASTExpression retValue=null]: {ASTExpression e2 = null;
ASTExpression e2tail = null;}
            #(EXPRESSION1 e2 = expression2 e2tail = expression2tail[e2])
            {
                retValue = e2tail;
            }
            ;

expression2tail[ASTExpression exprIn] returns [ASTExpression retValue=exprIn]:
{ASTExpression e2tail=null;  ASTExpression e2 = null;}
            #(tail:EXPRESSION2TAIL ((op1:ASSIGNEQUALOP | op2:PLUSEQUALOP |
op3:MINUSEQUALOP | op4:MULTEQUALOP | op5:DIVEQUALOP | op6:MODEQUALOP) e2 = expression2
            {
                if (exprIn.isConst())
                    sawError = CILYError.printError(exprIn.getValue() + " is a const
variable, and its value cannot be changed",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else if (!exprIn.isLvalue())
                    sawError = CILYError.printError(exprIn.getValue() + " cannot be assined
a value", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else
                {
                    retValue.setLvalue(false);
                    if (op1 != null && exprIn.getArrayDim() == 0)
                    {
                        e2.setOp(ASTOp.EQUALS);
                        retValue = exprIn.evaluate(e2,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op2 != null && exprIn.getArrayDim() == 0)
                    {
                        e2.setOp(ASTOp.PLUSEQUALS);
                        retValue = exprIn.evaluate(e2,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op3 != null && exprIn.getArrayDim() == 0)
                    {
                        e2.setOp(ASTOp.MINUSEQUALS);
                        retValue = exprIn.evaluate(e2,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op4 != null && exprIn.getArrayDim() == 0)
                    {
                        e2.setOp(ASTOp.MULTIPLYEQUALS);
                        retValue = exprIn.evaluate(e2,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                        if (debug) System.err.println("got to expression2tail -
assignment");
                    }
                    else if (op5 != null && exprIn.getArrayDim() == 0)
                    {
                        e2.setOp(ASTOp.DIVIDEEQUALS);
```

```
                    retValue = exprIn.evaluate(e2,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                    if (debug) System.err.println("got to expression2tail -
assignment");
                }
                else if (op6 != null && exprIn.getArrayDim() == 0)
                {
                    e2.setOp(ASTOp.MODEQUALS);
                    retValue = exprIn.evaluate(e2,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                    if (debug) System.err.println("got to expression2tail -
assignment");
                }
                else if (op1 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op1.getText() + "
with an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
                else if (op2 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op2.getText() + "
with an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
                else if (op3 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op3.getText() + "
with an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
                else if (op4 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op4.getText() + "
with an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
                else if (op5 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op5.getText() + "
with an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
                else if (op6 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op6.getText() + "
with an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
            }
        }
        e2tail = expression2tail[retValue])?)
        {
            if (e2tail != null)
                retValue = e2tail;
            if (debug) System.err.println("retvalue is " + retValue.getValue());
        }
        ;

expression2 returns [ASTExpression retValue=null]: {ASTExpression e3 = null;
ASTExpression e3tail = null;}
        #(EXPRESSION2 e3 = expression3 e3tail = expression3tail[e3])
        {
            retValue = e3tail;
        }
        ;

expression3tail[ASTExpression exprIn] returns [ASTExpression retValue=exprIn]:
{ASTExpression e3tail=null;  ASTExpression e3 = null;}
        #(tail:EXPRESSION3TAIL ((op1:RelationalOp | op2:EQUALEQUALOP | op3:NOTEQUALOP)
e3 = expression3
        {
            if (op1 != null && exprIn.getArrayDim() == 0)
            {
                retValue.setLvalue(false);
                if (op1.getText().equals(">"))
                    e3.setOp(ASTOp.GREATERTHAN);
                else if (op1.getText().equals("<"))
                    e3.setOp(ASTOp.LESSTHAN);
                else if (op1.getText().equals(">="))
                    e3.setOp(ASTOp.GREATEREQUALTHAN);
                else if (op1.getText().equals("<="))
                    e3.setOp(ASTOp.LESSEQUALTHAN);
```

```
                if (debug) System.err.println("got to expression3tail - relational");

                retValue = exprIn.evaluate(e3,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            }
            else if (op2 != null && exprIn.getArrayDim() == 0)
            {
                retValue.setLvalue(false);
                e3.setOp(ASTOp.EQUALEQUALS);
                if (debug) System.err.println("got to expression3tail - relational");
                retValue = exprIn.evaluate(e3,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            }
            else if (op3 != null && exprIn.getArrayDim() == 0)
            {
                retValue.setLvalue(false);
                e3.setOp(ASTOp.NOTEQUALS);
                retValue = exprIn.evaluate(e3,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                if (debug) System.err.println("got to expression3tail - relational");
            }
            else if (op1 != null && exprIn.getArrayDim() != 0)
                sawError = CILYError.printError("Cannot use " + op1.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            else if (op2 != null && exprIn.getArrayDim() != 0)
                sawError = CILYError.printError("Cannot use " + op2.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            else if (op3 != null && exprIn.getArrayDim() != 0)
                sawError = CILYError.printError("Cannot use " + op3.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            }
            e3tail = expression3tail[retValue])?)
            {
                if (e3tail != null)
                    retValue = e3tail;
                if (debug) System.err.println("retvalue is " + retValue.getValue());
            }

            ;

expression3 returns [ASTExpression retValue=null]: {ASTExpression e4 = null;
ASTExpression e4tail = null;}

            #(EXPRESSION3 e4 = expression4 e4tail = expression4tail[e4])
            {
                retValue = e4tail;
                if (debug) System.err.println("made it to expression 3 right before
evaluate");
            }
            ;

expression4tail[ASTExpression exprIn] returns [ASTExpression retValue=exprIn]:
{ASTExpression e4tail=null;  ASTExpression e4 = null;}
            #(tail:EXPRESSION4TAIL ((op1:PLUSOP | op2:MINUSOP) e4 = expression4
            {
                if (op1 != null && exprIn.getArrayDim() == 0)
                {
                    retValue.setLvalue(false);
                    e4.setOp(ASTOp.PLUS);
                    retValue = exprIn.evaluate(e4,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));

                    if (debug) System.err.println("got to expression4tail - addsub " +
retValue.getValue());
                }
                else if (op2 != null && exprIn.getArrayDim() == 0)
                {
                    retValue.setLvalue(false);
                    e4.setOp(ASTOp.MINUS);
```

```
                    retValue = exprIn.evaluate(e4,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));  //retValue
op is set in evaluate

                    if (debug) System.err.println("got to expression4tail - addsub");
                }
                else if (op1 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op1.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else if (op2 != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op2.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            }
            e4tail = expression4tail[retValue])?)
            {
                if (e4tail != null)
                    retValue = e4tail;
                if (debug) System.err.println("retvalue is " + retValue.getValue());
            }
            ;

expression4 returns [ASTExpression retValue=null]: {ASTExpression e5 = null;
ASTExpression e5tail = null;}

            #(EXPRESSION4 e5 = expression5 e5tail = expression5tail[e5])
            {
                retValue = e5tail;
            }
            ;

expression5tail[ASTExpression exprIn] returns [ASTExpression retValue=exprIn]:
{ASTExpression e5tail=null;  ASTExpression e5 = null;}
            #(tail:EXPRESSION5TAIL (op:ArithmeticOp e5 = expression5
            {
                if (op != null && exprIn.getArrayDim() == 0)
                {
                    retValue.setLvalue(false);
                    if (op.getText().equals("*"))
                        e5.setOp(ASTOp.MULTIPLY);
                    else if (op.getText().equals("/"))
                        e5.setOp(ASTOp.DIVIDE);  // a / 3 * 8 = a * (8/3)
                    else if (op.getText().equals("%"))
                        e5.setOp(ASTOp.MOD);
                    if (debug) System.err.println("got to expression5tail - multdivmod");

                    retValue = exprIn.evaluate(e5,
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                }
                else if (op != null && exprIn.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot use " + op.getText() + " with
an array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            }
            e5tail = expression5tail[retValue])?)
            {
                if (e5tail != null)
                    retValue = e5tail;
                if (debug) System.err.println("retvalue is " + retValue.getValue());
            }
            ;

expression5 returns [ASTExpression retValue=null]: {ASTExpression expr5;}
            #(exp:EXPRESSION5 (op1:PLUSOP | op2:MINUSOP | op3:NOT)? expr5 = rvalue)
            {
                retValue = expr5;
                if (op1 != null)
                {
                    retValue.setLvalue(false);
                    if (debug) System.err.println("got to expression5 - plusop");
                    if ((expr5.getType() !=  ASTType.DOUBLE && expr5.getType() !=
ASTType.INT) || expr5.getArrayDim() != 0)
```

```
                    sawError = CILYError.printError("Cannot put \'+\' in front of a " +
ASTType.getStringValue(expr5.getType()),
Integer.parseInt(exp.getText().substring(exp.getText().indexOf(':') + 1)));
                }
                else if (op2 != null) // found negation of expression
                {
                    retValue.setLvalue(false);
                    if (debug) System.err.println("got to expression5 - minusop");
                    if ((expr5.getType() !=  ASTType.DOUBLE && expr5.getType() !=
ASTType.INT) || expr5.getArrayDim() != 0)
                    sawError = CILYError.printError("Cannot put \'-\' in front of a " +
ASTType.getStringValue(expr5.getType()),
Integer.parseInt(exp.getText().substring(exp.getText().indexOf(':') + 1)));
                    else
                    {
                        if (expr5.getType() == ASTType.DOUBLE)
                        {
                            double v = -((Double)expr5.getValue()).doubleValue();
                            retValue.setValue(new Double(v));
                        }
                        else if (expr5.getType() == ASTType.INT)
                        {
                            int v = -((Integer)expr5.getValue()).intValue();
                            retValue.setValue(new Integer(v));
                        }
                    }
                }
                else if (op3 != null)
                {
                    retValue.setLvalue(false);
                    if (debug) System.err.println("got to expression5 - not");
                    if (expr5.getType() !=  ASTType.BOOLEAN || expr5.getArrayDim() != 0)
                        sawError = CILYError.printError("Cannot put \'!\' in front of a " +
ASTType.getStringValue(expr5.getType()),
Integer.parseInt(exp.getText().substring(exp.getText().indexOf(':') + 1)));
                    else
                    {
                        boolean v = !((Boolean)expr5.getValue()).booleanValue();
                        retValue.setValue(new Boolean(v));
                    }
                }
            }
            ;

rvalue returns [ASTExpression retValue=null]: {ASTExpression cnstExpr=null, parenExpr =
null, lval = null, func = null;}
            #(RVALUE (lval = lvalue | parenExpr = expression | func = functioncall |
cnstExpr = constant))
            {
                retValue = null;
                if (lval != null)
                {
                    retValue = lval;
                    if (debug) System.err.println("got to rvalue - lvalue");
                }
                else if (parenExpr != null)
                {
                    retValue = parenExpr;
                    retValue.setLvalue(false);
                    if (!retValue.getEvaluated())
                        retValue.setValue(new String("(" + retValue.getValue() + ")"));
                    if (debug) System.err.println("got to rvalue - expression");
                }
                else if (func != null)
                {
                    retValue = func;
                    if (debug) System.err.println("got to rvalue - functioncall");
                }
                else if (cnstExpr != null)
                {
                    retValue = cnstExpr;
```

```
                if (debug) System.err.println("Got to rvalue - constexpr");
            }

        }
        ;

lvalue returns [ASTExpression retValue=null]: {ASTExpression lvalueExpr=null,
lvaltail=null;}
        #(left:LVALUE name:Id
        {
            int type = ASTType.VOID;
            boolean isConst = false;
            boolean isLvalue = false;
            int arrayDim = 0;
            if (debug) System.err.println("Got here to lvalue = id...");
            // if returns -1, no id exists in the symbol table
            Object obj = currentASTScope.withinScope(name.getText());
            if (obj == null)
                sawError = CILYError.printError("undeclared identifier " +
name.getText(), Integer.parseInt(left.getText().substring(left.getText().indexOf(':') +
1)));
            else
            {
                if (obj.getClass().getName().equals("ASTVariable"))
                {
                    type = ((ASTVariable)obj).getType();
                    isLvalue = true;
                    if (((ASTVariable)obj).isConst())
                        isConst = true;
                }
                else if (obj.getClass().getName().equals("ASTMyArray"))
                {
                    isLvalue = true;
                    type = ((ASTMyArray)obj).getType();
                    arrayDim = ((ASTMyArray)obj).getNumDimension();
                }
            }
            retValue = new ASTExpression(type, new String(name.getText()), false,
arrayDim);
            retValue.setConst(isConst);
            retValue.setLvalue(isLvalue);
        }
        lvaltail = lvaluetail[retValue])
        {
            if (debug) System.err.println("got to lvalue id = " + name.getText());
            if (lvaltail != null)
                retValue = lvaltail;
        }
        ;

lvaluetail [ASTExpression lval] returns [ASTExpression retValue=lval]: {ASTExpression
expr1=null, expr2=null;}
        #(tail:LVALUETAIL ((expr1 = expression (expr2 = expression)?)
        {
            Object obj = currentASTScope.withinScope(new
String(retValue.getValue().toString()));
            if (retValue.getArrayDim() == 0)
                sawError = CILYError.printError("array dimension does not match the
declared dimension.",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
            else
            {
                if (obj.getClass().getName().equals("ASTMyArray"))
                {
                    if (expr1.getType() == ASTType.INT)
                    {
                        retValue.setValue(retValue.getValue() + "[" + expr1.getValue()
+ "]");
                        retValue.setArrayDim(retValue.getArrayDim() - 1);
                    }
                    else
```

```
                        sawError = CILYError.printError("array index must be non-
negative integer", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':')
+ 1)));
                    if (expr2 == null && ((ASTMyArray)obj).getNumDimension() == 2)
                        sawError = CILYError.printError("two indeces must be specified
for a two dimensional array",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                    else if (expr2 != null && expr2.getType() == ASTType.INT)
                    {
                        if (((ASTMyArray)obj).getNumDimension() == 2)
                        {
                            retValue.setValue(retValue.getValue() + "[" +
expr2.getValue() + "]");
                            retValue.setArrayDim(retValue.getArrayDim() - 1);
                        }
                        else
                            sawError = CILYError.printError("array is not a two-
dimensional array", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':')
+ 1)));
                    }
                    else if (expr2 != null && expr2.getType() != ASTType.INT)
                        sawError = CILYError.printError("array index must be non-
negative integer", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':')
+ 1)));
                }
            }
        }
        retValue = lvaluetailtail[retValue]
        | retValue = lvaluetailtail[retValue])
        )
        ;

lvaluetailtail [ASTExpression lval] returns [ASTExpression retValue=lval]: {ASTExpression
expr1=null, expr2=null, expr=null;}
        #(tail:LVALUETAILTAIL (op1:PLUSPLUSOP | op2:MINUSMINUSOP | (arrow:ARROWOP
name:Id
        {
            if (arrow != null)
            {
                int type = ASTType.checkSubType(lval.getType(), name.getText());
                if (type < 0)
                    sawError = CILYError.printError("cannot find member " +
name.getText(), Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') +
1)));
                else if (lval.getType() == ASTType.IMAGE &&
name.getText().equals("width"))
                {
                    retValue = new ASTExpression(type, new String(lval.getValue() +
".bi.getWidth(null)"), false);
                    retValue.setLvalue(false);
                }
                else if (lval.getType() == ASTType.IMAGE &&
name.getText().equals("height"))
                {
                    retValue = new ASTExpression(type, new String(lval.getValue() +
".bi.getHeight(null)"), false);
                    retValue.setLvalue(false);
                }
                else
                {
                    retValue = new ASTExpression(type, new String(lval.getValue() + "."
+ name.getText()), false);
                    retValue.setLvalue(true);
                }
            }
        }
        retValue = lvaluetailtail[retValue])
        )?
        {
            if (op1 != null || op2 != null || arrow != null || expr1 != null ||
expr2 != null)
```

```
                if (debug) System.err.println("got to lvaluetailtail");

            if (op1 != null)
            {
                if (retValue.getType() != ASTType.INT)
                    sawError = CILYError.printError("cannot use \"++\" with non-
integers", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else if (retValue.getArrayDim() != 0)
                    sawError = CILYError.printError("cannot use \"++\" with an array",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else if (!retValue.isLvalue())
                    sawError = CILYError.printError("cannot use \"++\" with the left
hand side of thie expression",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else
                {
                    retValue.setValue(lval.getValue() + " = " + retValue.getValue() + "
+ 1");
                    retValue.setUnaryExpression(true);
                    retValue.setLvalue(false);
                }
            }
            else if (op2 != null)
            {
                if (retValue.getType() != ASTType.INT)
                    sawError = CILYError.printError("cannot use \"--\" with non-
integers", Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else if (retValue.getArrayDim() != 0)
                    sawError = CILYError.printError("cannot use \"--\" with an array",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else if (!retValue.isLvalue())
                    sawError = CILYError.printError("cannot use \"++\" with the left
hand side of thie expression",
Integer.parseInt(tail.getText().substring(tail.getText().indexOf(':') + 1)));
                else
                {
                    retValue.setValue(retValue.getValue() + " = " + retValue.getValue()
+ " - 1");
                    retValue.setUnaryExpression(true);
                    retValue.setLvalue(false);
                }
            }
        }
        )
        ;

constant returns [ASTExpression retValue=null]:
        #(CONSTANT (num:Number | str:StringConstant | bool:BOOLEANCONSTANT))
        {
            if (num != null)
            {
                if (debug) System.err.println("got to constant: num = " +
num.getText());
                if (num.getText().indexOf(".") >= 0)
                    return new ASTExpression(ASTType.DOUBLE, new Double(num.getText()),
true);
                else
                    return new ASTExpression(ASTType.INT, new Integer(num.getText()),
true);
            }
            else if (str != null)
            {
                String s = str.getText().replaceAll("\"", "\\\\\"");
                if (debug) System.err.println("got to constant: str = " +
str.getText());
                return new ASTExpression(ASTType.STRING, s, true);
            }
            else if (bool != null)
            {
                if (debug) System.err.println("got to constant: bool = " +
bool.getText());
```

```
                    return new ASTExpression(ASTType.BOOLEAN, new Boolean(bool.getText()),
true);
                }
            }
            ;
```

```
// ASTArrayInfo.java
// Authors: Makiko Yasui

public class ASTArrayInfo
{
    private String value;
    private int size1;
    private int size2;

    public ASTArrayInfo()
    {
        value = "";
        size1 = 0;
        size2 = 0;
    }

    public void incrementSize1()
    {
        size1++;
    }

    public void incrementSize2()
    {
        size2++;
    }

    public void setSize1(int s)
    {
        size1 = s;
    }

    public void setSize2(int s)
    {
        size2 = s;
    }

    public void setValue(String v)
    {
        value = v;
    }

    public void addValue(String v)
    {
        value += v;
    }

    public int getSize1()
    {
        return size1;
    }

    public int getSize2()
    {
        return size2;
    }

    public String getValue()
    {
        return value;
    }
}
```

```
// ASTExpression.java
// Authors: Makiko Yasui, Edward Ishak
public class ASTExpression
{
    private int type;
    private Object value;
    private boolean evaluated = true;
    private int op;
    private boolean unaryExpression = false;
    private int arrayDim = 0;
    private boolean isConst = false;
    private boolean isLvalue = false;

    public ASTExpression(int type, Object value)
    {
        this.type = type;
        this.value = value;
        this.op = -1;
        this.evaluated = true;
        this.unaryExpression = false;
        this.arrayDim = 0;
        this.isConst = false;
        this.isLvalue = false;
    }

    public ASTExpression(int type, Object value, int op)
    {
        this.type = type;
        this.value = value;
        this.op = op;
        this.evaluated = false;
        this.unaryExpression = false;
        this.arrayDim = 0;
        this.isConst = false;
        this.isLvalue = false;
    }

    public ASTExpression(int type, Object value, boolean eval)
    {
        this.type = type;
        this.value = value;
        this.evaluated = eval;
        this.op = -1;
        this.unaryExpression = false;
        this.arrayDim = 0;
        this.isConst = false;
        this.isLvalue = false;
    }

    public ASTExpression(int type, Object value, boolean eval, int a)
    {
        this.type = type;
        this.value = value;
        this.evaluated = eval;
        this.op = -1;
        this.unaryExpression = false;
        this.arrayDim = a;
        this.isConst = false;
        this.isLvalue = false;
    }

    public ASTExpression(ASTExpression expr)
    {
        this.type = expr.getType();
        this.value = expr.getValue();
        this.evaluated = expr.getEvaluated();
        this.op = expr.getOp();
        this.unaryExpression = expr.isUnaryExpression();
        this.arrayDim = expr.getArrayDim();
        this.isConst = expr.isConst();
        this.isLvalue = expr.isLvalue();
```

```
}

public int getType()
{
    return type;
}

public int getArrayDim()
{
    return arrayDim;
}

public Object getValue()
{
    return value;
}

public boolean getEvaluated()
{
    return evaluated;
}

public boolean isUnaryExpression()
{
    return unaryExpression;
}

public boolean isConst()
{
    return isConst;
}

public boolean isLvalue()
{
    return isLvalue;
}

public int getOp()
{
    return op;
}

public void setType(int type)
{
    this.type = type;
}

public void setArrayDim(int dim)
{
    arrayDim = dim;
}

public void setValue(Object value)
{
    this.value = value;
}

public void setConst(boolean c)
{
    this.isConst = c;
}

public void setLvalue(boolean c)
{
    this.isLvalue = c;
}

public void setOp(int op)
{
    this.op = op;
}
```

```
    public void setEvaluated(boolean eval)
    {
        this.evaluated = eval;
    }

    public void setUnaryExpression(boolean unary)
    {
        this.unaryExpression = unary;
    }

    public ASTExpression evaluate(ASTExpression rhs, int line)
    {
        int currentType = this.type;
        ASTExpression retValue = new ASTExpression(this);

        if (rhs == null)
        {
            return retValue;
        }

        if (this.type != rhs.getType())
        {
            // if both are not strings    OR    if one is double and other is NOT int   OR
if one is int and other is NOT double
            if (!((this.type == ASTType.STRING || rhs.getType() == ASTType.STRING) ||
                (this.type == ASTType.DOUBLE && (rhs.getType() == ASTType.STRING ||
rhs.getType() == ASTType.INT || rhs.getType() == ASTType.DOUBLE))
                || (this.type == ASTType.INT && (rhs.getType() == ASTType.STRING ||
rhs.getType() == ASTType.DOUBLE || rhs.getType() == ASTType.INT))))
            {
                CILYError.printError("cannot use " + ASTOp.getStringValue(rhs.getOp()) + "
with " + ASTType.getStringValue(this.type) + " and " +
ASTType.getStringValue(rhs.getType()), line);
                return retValue;
            }
        }

        // we have compatible types, now check compatible ops
        if (this.type == ASTType.STRING || rhs.getType() == ASTType.STRING)
        {
            currentType = ASTType.STRING;
        }
        else if (this.type == ASTType.DOUBLE || rhs.getType() == ASTType.DOUBLE)
        {
            retValue.setType(ASTType.DOUBLE);
            currentType = ASTType.DOUBLE;
        }

        if (!evaluated || !rhs.getEvaluated())
        {
            if (rhs.getOp() == ASTOp.EQUALEQUALS || rhs.getOp() == ASTOp.NOTEQUALS ||
rhs.getOp() == ASTOp.GREATERTHAN ||
                rhs.getOp() == ASTOp.LESSTHAN || rhs.getOp() == ASTOp.GREATEREQUALTHAN ||
rhs.getOp() == ASTOp.LESSEQUALTHAN ||
                rhs.getOp() == ASTOp.AND || rhs.getOp() == ASTOp.OR)
            {
                if (rhs.getOp() == ASTOp.EQUALEQUALS && currentType == ASTType.STRING)
                {
                    if (!evaluated)
                    {
                        if (!rhs.getEvaluated() && rhs.getType() == ASTType.STRING &&
this.type == ASTType.STRING)
                        {
                            retValue.setValue(new String(retValue.getValue() + ".equals(" +
rhs.getValue() + ")"));
                        }
                        else if (rhs.getEvaluated() && rhs.getType() == ASTType.STRING &&
this.type == ASTType.STRING)
                        {
```

```
                        retValue.setValue(new String(retValue.getValue() + ".equals(\""
+ rhs.getValue() + "\")"));
                    }
                    else if (rhs.getType() == ASTType.INT)
                    {
                        retValue.setValue(new String(retValue.getValue() +
".equals((new Integer(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (rhs.getType() == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String(retValue.getValue() +
".equals((new Double(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (this.type == ASTType.INT)
                    {
                        retValue.setValue(new String("(new Integer(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                    else if (this.type == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("(new Double(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                }
                else //lhs is evaluated so is a constant
                {
                    if (rhs.getType() == ASTType.STRING && this.type == ASTType.STRING)
                    {
                        retValue.setValue(new String("\"" + retValue.getValue() +
"\".equals(" + rhs.getValue() + ")"));
                    }
                    else if (rhs.getType() == ASTType.INT)
                    {
                        retValue.setValue(new String("\"" + retValue.getValue() +
"\".equals((new Integer(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (rhs.getType() == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("\"" + retValue.getValue() +
"\".equals((new Double(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (this.type == ASTType.INT)
                    {
                        retValue.setValue(new String("(new Integer(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                    else if (this.type == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("(new Double(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                }
            }
            else if (rhs.getOp() == ASTOp.NOTEQUALS && currentType == ASTType.STRING)
            {
                if (!evaluated)
                {
                    if (!rhs.getEvaluated() && rhs.getType() == ASTType.STRING &&
this.type == ASTType.STRING)
                    {
                        retValue.setValue(new String("!" + retValue.getValue() +
".equals(" + rhs.getValue() + ")"));
                    }
                    else if (rhs.getEvaluated() && rhs.getType() == ASTType.STRING &&
this.type == ASTType.STRING)
                    {
                        retValue.setValue(new String("!" + retValue.getValue() +
".equals(\"" + rhs.getValue() + "\")"));
                    }
                    else if (rhs.getType() == ASTType.INT)
                    {
```

```
                        retValue.setValue(new String("!" + retValue.getValue() +
".equals((new Integer(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (rhs.getType() == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("!" + retValue.getValue() +
".equals((new Double(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (this.type == ASTType.INT)
                    {
                        retValue.setValue(new String("!((new Integer(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                    else if (this.type == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("!((new Double(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                }
                else //lhs is evaluated so is a constant
                {
                    if (rhs.getType() == ASTType.STRING && this.type == ASTType.STRING)
                    {
                        retValue.setValue(new String("!\"" + retValue.getValue() +
"\".equals(" + rhs.getValue() + ")"));
                    }
                    else if (rhs.getType() == ASTType.INT)
                    {
                        retValue.setValue(new String("!\"" + retValue.getValue() +
"\".equals((new Integer(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (rhs.getType() == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("!\"" + retValue.getValue() +
"\".equals((new Double(" + rhs.getValue() + ")).toString())"));
                    }
                    else if (this.type == ASTType.INT)
                    {
                        retValue.setValue(new String("!((new Integer(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                    else if (this.type == ASTType.DOUBLE)
                    {
                        retValue.setValue(new String("!((new Double(" +
retValue.getValue() + ")).toString().equals(" + rhs.getValue() + ")"));
                    }
                }
            }
            else if ((rhs.getOp() == ASTOp.GREATERTHAN || rhs.getOp() ==
ASTOp.LESSTHAN || rhs.getOp() == ASTOp.GREATEREQUALTHAN || rhs.getOp() ==
ASTOp.LESSEQUALTHAN)
                && (currentType == ASTType.INT || currentType == ASTType.DOUBLE))
            {
                retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
            }
            else if ((rhs.getOp() == ASTOp.EQUALEQUALS || rhs.getOp() ==
ASTOp.NOTEQUALS) && (currentType == ASTType.INT || currentType == ASTType.DOUBLE ||
currentType == ASTType.BOOLEAN))
            {
                retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
            }
            else if ((rhs.getOp() == ASTOp.AND || rhs.getOp() == ASTOp.OR) &&
currentType == ASTType.BOOLEAN)
            {
                retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
            }
            else
            {
```

```
                CILYError.printError("cannot use " + ASTOp.getStringValue(rhs.getOp())
+ " with " + ASTType.getStringValue(rhs.getType()) + " and " +
ASTType.getStringValue(this.type), line);
            }
            retValue.setType(ASTType.BOOLEAN);
            retValue.setEvaluated(false);
        }
        else
        {
            if (rhs.getOp() == ASTOp.PLUS && currentType == ASTType.STRING)
            {
                if (!evaluated && !rhs.getEvaluated())
                {
                    retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
                }
                else if (!evaluated)
                {
                    retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "\"" + rhs.getValue() + "\""));
                }
                else
                {
                    retValue.setValue(new String("\"" + retValue.getValue() + "\"" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
                }
                retValue.setType(ASTType.STRING);
            }
            else if ((rhs.getOp() == ASTOp.EQUALS || rhs.getOp() == ASTOp.PLUSEQUALS)
&& this.type == ASTType.STRING)
            {
                retValue.setType(ASTType.STRING);
                if (!rhs.getEvaluated())
                {
                    retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
                }
                else
                {
                    retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "\"" + rhs.getValue() + "\""));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE &&
(rhs.getOp() == ASTOp.EQUALS || rhs.getOp() == ASTOp.PLUSEQUALS || rhs.getOp() ==
ASTOp.MINUSEQUALS || rhs.getOp() == ASTOp.MULTIPLYEQUALS || rhs.getOp() ==
ASTOp.DIVIDEEQUALS || rhs.getOp() == ASTOp.MODEQUALS))
            {
                retValue.setType(ASTType.INT);
                retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "(int)(" + rhs.getValue() + ")"));
            }
            else if (currentType == ASTType.INT || currentType == ASTType.DOUBLE)
            {
                retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
            }
            else if (rhs.getOp() == ASTOp.EQUALS && rhs.getType() == this.type)
            {
                retValue.setValue(new String(retValue.getValue() + "" +
ASTOp.getStringValue(rhs.getOp()) + "" + rhs.getValue() + ""));
            }
            else
            {
                CILYError.printError("cannot use " + ASTOp.getStringValue(rhs.getOp())
+ " with " + ASTType.getStringValue(rhs.getType()) + " and " +
ASTType.getStringValue(this.type), line);
            }
            retValue.setEvaluated(false);
        }
        return retValue;
```

```
        }

        if (rhs.getOp() == ASTOp.PLUS)
        {
            int i = 1;
            if (this.op == ASTOp.MINUS)
            {
                i = -1;
            }
            if (currentType == ASTType.STRING)
            {
                retValue = new ASTExpression(ASTType.STRING, new
String(this.value.toString() + rhs.getValue().toString()));
            }
            else if (currentType == ASTType.INT)
            {
                int newValue = ((Integer)this.value).intValue() * i +
((Integer)rhs.getValue()).intValue();
                retValue = new ASTExpression(ASTType.INT, new Integer(newValue));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                double newValue = ((Double)this.value).doubleValue() * i +
((Double)rhs.getValue()).doubleValue();
                retValue = new ASTExpression(ASTType.DOUBLE, new Double(newValue));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                double newValue = ((Double)this.value).doubleValue() * i +
(double)(((Integer)rhs.getValue()).intValue());
                retValue = new ASTExpression(ASTType.DOUBLE, new Double(newValue));
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                double newValue = (double)(((Integer)this.value).intValue()) * i +
((Double)rhs.getValue()).doubleValue();
                retValue = new ASTExpression(ASTType.DOUBLE, new Double(newValue));
            }
            else
            {
                CILYError.printError("cannot use \"+\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.MINUS)
        {
            int i = 1;
            if (this.op == ASTOp.MINUS)
            {
                i = -1;
            }
            if (currentType == ASTType.INT)
            {
                int newValue = ((Integer)this.value).intValue() * i -
((Integer)rhs.getValue()).intValue();
                retValue = new ASTExpression(ASTType.INT, new Integer(newValue));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                double newValue = ((Double)this.value).doubleValue() * i -
((Double)rhs.getValue()).doubleValue();
                retValue = new ASTExpression(ASTType.DOUBLE, new Double(newValue));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                double newValue = ((Double)this.value).doubleValue() * i -
(double)(((Integer)rhs.getValue()).intValue());
                retValue = new ASTExpression(ASTType.DOUBLE, new Double(newValue));
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
```

```
            {
                double newValue = (double)(((Integer)this.value).intValue()) * i -
((Double)rhs.getValue()).doubleValue();
                retValue = new ASTExpression(ASTType.DOUBLE, new Double(newValue));
            }
            else
            {
                CILYError.printError("cannot use \"-\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.MULTIPLY)
        {
            if (currentType == ASTType.INT)
            {
                retValue = new ASTExpression(ASTType.INT, new
Integer(((Integer)this.value).intValue() * ((Integer)rhs.getValue()).intValue()));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                retValue = new ASTExpression(ASTType.DOUBLE, new
Double(((Double)this.value).doubleValue() * ((Double)rhs.getValue()).doubleValue()));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                retValue = new ASTExpression(ASTType.DOUBLE, new
Double(((Double)this.value).doubleValue() *
(double)(((Integer)rhs.getValue()).intValue())));
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                retValue = new ASTExpression(ASTType.DOUBLE, new
Double((double)(((Integer)this.value).intValue()) *
((Double)rhs.getValue()).doubleValue()));
            }
            else
            {
                CILYError.printError("cannot use \"*\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.DIVIDE)
        {
            if (currentType == ASTType.INT)
            {
                retValue = new ASTExpression(ASTType.INT, new
Integer(((Integer)this.value).intValue() / ((Integer)rhs.getValue()).intValue()));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                retValue = new ASTExpression(ASTType.DOUBLE, new
Double(((Double)this.value).doubleValue() / ((Double)rhs.getValue()).doubleValue()));
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                retValue = new ASTExpression(ASTType.DOUBLE, new
Double(((Double)this.value).doubleValue() /
(double)(((Integer)rhs.getValue()).intValue())));
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                retValue = new ASTExpression(ASTType.DOUBLE, new
Double((double)(((Integer)this.value).intValue()) /
((Double)rhs.getValue()).doubleValue()));
            }
            else
            {
```

```
                CILYError.printError("cannot use \"/\" " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.MOD)
        {
            if (currentType == ASTType.INT)
            {
                retValue = new ASTExpression(ASTType.INT, new
Integer(((Integer)this.value).intValue() % ((Integer)rhs.getValue()).intValue()));
            }
            else
            {
                    CILYError.printError("cannot use \"%\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.EQUALEQUALS)
        {
            if (currentType == ASTType.INT)
            {
                if (((Integer)this.value).intValue() ==
((Integer)rhs.getValue()).intValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                if (((Double)this.value).doubleValue() ==
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                if ((double)((Integer)this.value).doubleValue() ==
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                if (((Double)this.value).doubleValue() ==
(double)(((Integer)rhs.getValue()).intValue()))
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (currentType == ASTType.BOOLEAN)
            {
```

```
                if (((Boolean)this.value).booleanValue() ==
((Boolean)rhs.getValue()).booleanValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (currentType == ASTType.STRING)
            {
                if ((this.type == ASTType.STRING && (rhs.getType() == ASTType.STRING ||
rhs.getType() == ASTType.INT || rhs.getType() == ASTType.DOUBLE))
                    || (rhs.getType() == ASTType.STRING && (this.type == ASTType.INT ||
this.type == ASTType.DOUBLE)))
                {
                    if (this.value.toString().equals(rhs.getValue().toString()))
                    {
                        retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                    }
                    else
                    {
                        retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                    }
                }
            }
            else
            {
                    CILYError.printError("cannot use \"==\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.NOTEQUALS)
        {
            if (currentType == ASTType.INT)
            {
                if (((Integer)this.value).intValue() !=
((Integer)rhs.getValue()).intValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                if (((Double)this.value).doubleValue() !=
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                if ((double)(((Integer)this.value).intValue()) !=
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
```

```
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                if (((Double)this.value).doubleValue() !=
(double)(((Integer)rhs.getValue()).intValue()))
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (currentType == ASTType.BOOLEAN)
            {
                if (((Boolean)this.value).booleanValue() !=
((Boolean)rhs.getValue()).booleanValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (currentType == ASTType.STRING)
            {
                if ((this.type == ASTType.STRING && (rhs.getType() == ASTType.STRING ||
rhs.getType() == ASTType.INT || rhs.getType() == ASTType.DOUBLE))
                    || (rhs.getType() == ASTType.STRING && (this.type == ASTType.INT ||
this.type == ASTType.DOUBLE)))
                {
                    if (!this.value.toString().equals(rhs.getValue().toString()))
                    {
                        retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                    }
                    else
                    {
                        retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                    }
                }
            }
            else
            {
                    CILYError.printError("cannot use \"!=\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.AND)
        {
            if (currentType == ASTType.BOOLEAN)
            {
                if (((Boolean)this.value).booleanValue() &&
((Boolean)rhs.getValue()).booleanValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else
            {
                    CILYError.printError("cannot use \"&&\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.OR)
```

```
        {
            if (currentType == ASTType.BOOLEAN)
            {
                if (((Boolean)this.value).booleanValue() ||
((Boolean)rhs.getValue()).booleanValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else
            {
                    CILYError.printError("cannot use \"||\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.GREATERTHAN)
        {
            if (currentType == ASTType.INT)
            {
                if (((Integer)this.value).intValue() >
((Integer)rhs.getValue()).intValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                if (((Double)this.value).doubleValue() >
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                if ((double)((Integer)this.value).doubleValue() >
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                if (((Double)this.value).doubleValue() >
(double)(((Integer)rhs.getValue()).intValue()))
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else
```

```
            {
                    CILYError.printError("cannot use \">\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.GREATEREQUALTHAN)
        {
            if (currentType == ASTType.INT)
            {
                if (((Integer)this.value).intValue() >=
((Integer)rhs.getValue()).intValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                if (((Double)this.value).doubleValue() >=
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                if ((double)(((Integer)this.value).intValue()) >=
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                if (((Double)this.value).doubleValue() >=
(double)(((Integer)rhs.getValue()).intValue()))
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else
            {
                    CILYError.printError("cannot use \">=\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.LESSTHAN)
        {
            if (currentType == ASTType.INT)
            {
                if (((Integer)this.value).intValue() <
((Integer)rhs.getValue()).intValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
```

```
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                if (((Double)this.value).doubleValue() <
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                if ((double)(((Integer)this.value).intValue()) <
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                if (((Double)this.value).doubleValue() <
(double)((Integer)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else
            {
                    CILYError.printError("cannot use \"<\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        else if (rhs.getOp() == ASTOp.LESSEQUALTHAN)
        {
            if (currentType == ASTType.INT)
            {
                if (((Integer)this.value).intValue() <=
((Integer)rhs.getValue()).intValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.DOUBLE)
            {
                if (((Double)this.value).doubleValue() <=
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
```

```
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.INT && rhs.getType() == ASTType.DOUBLE)
            {
                if ((double)(((Integer)this.value).intValue()) <=
((Double)rhs.getValue()).doubleValue())
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else if (this.type == ASTType.DOUBLE && rhs.getType() == ASTType.INT)
            {
                if (((Double)this.value).doubleValue() <=
(double)(((Integer)rhs.getValue()).intValue()))
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(true));
                }
                else
                {
                    retValue = new ASTExpression(ASTType.BOOLEAN, new Boolean(false));
                }
            }
            else
            {
                    CILYError.printError("cannot use \"<=\" with " +
ASTType.getStringValue(rhs.getType()) + " and " + ASTType.getStringValue(this.type),
line);
            }
        }
        return retValue;
    }
}
```

```
// ASTFunction.java
// Authors: Makiko Yasui, Edward Ishak
import java.util.ArrayList;

public class ASTFunction extends ASTScope
{
    private int type;
    private String name;
    private ArrayList arguments;
    private boolean isLibraryFunction;
    private int arrayDim;

    public ASTFunction(String t, String n, ArrayList args, ASTScope p, boolean
libraryFunc)
    {
        super(p, 1);

        type = ASTType.getTypeValue(t);
        name = n;
        arguments = args;
        isLibraryFunction = libraryFunc;
        arrayDim = 0;
    }

   public ASTFunction(String t, String n, ArrayList args, ASTScope p, boolean libraryFunc,
int a)
    {
        super(p, 1);

        type = ASTType.getTypeValue(t);
        name = n;
        arguments = args;
        isLibraryFunction = libraryFunc;
        arrayDim = a;
    }

    public ASTFunction(ASTFunction f)
    {
        super(f.getParent(), 1);
        type = f.getReturnType();
        name = f.getName();
        arguments = f.getArgs();
        isLibraryFunction = f.isLibraryFunction;
        arrayDim = f.getArrayDim();
    }

    public int getReturnType()
    {
        return type;
    }

    public int getArrayDim()
    {
        return arrayDim;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String n)
    {
        name = n;
    }

    public void setReturnType(int a)
    {
        arrayDim = a;
    }
```

```
    public boolean isLibraryFunction()
    {
        return isLibraryFunction;
    }

    public ASTScope getParent()
    {
        return parent;
    }

    public ArrayList getArgs()
    {
        return arguments;
    }

    public ASTVariable getArg(int i)
    {
        if (i < arguments.size())
            return (ASTVariable)arguments.get(i);
        else
            return null;
    }

    public int getArgDim(int i)
    {
        if (i < arguments.size())
            return ((ASTVariable)arguments.get(i)).getArrayDim();
        else
            return -1;
    }

    public int getArgSize()
    {
            return arguments.size();
    }

    public String toString()
    {
        String str = "function: ";
        str += "type = " + ASTType.getStringValue(type);

        str += ", name = " + name;
        for (int i = 0; i < arguments.size(); i++)
        {
            str += getArg(i).toString();
        }
        str += "\n\t" + super.toString();
        return str;
    }
}
```

```
// ASTMyArray.java
// Authors: Makiko Yasui, Edward Ishak
import java.util.ArrayList;

public class ASTMyArray extends ASTVariable
{
//  private ASTExpression[] length;
    private int numDimension;

    public ASTMyArray(String t, String n, boolean b, int numD)
    {
        super(t, n, null, b);
        numDimension = numD;
//      length = size;
/*
        if (numDimensions == 2)
        {
            for (int i = 0; i < length[0]; i++)
                array.add(new ArrayList());
        }
*/
    }

    public int getNumDimension()
    {
        return numDimension;
    }
/*
    public ASTExpression getLength(int i)
    {
        return length[i];
    }

    public Object getValue(int index)
    {
        return array.get(index);
    }

    public Object getValue(int index1, int index2)
    {
        return ((ArrayList)array.get(index1)).get(index2);
    }

    public void setValue(int index, Object value)
    {
        array.set(index, value);
    }

    public void setValue(int index1, int index2, Object value)
    {
        array.set(index1, ((ArrayList)array.get(index1)).set(index2, value));
    }
*/
    public String toString()
    {
        String str = "array: " + super.toString();
        str += ", numD = " + numDimension;
        return str;
    }
}
```

```
// ASTOp.java
// Authors: Makiko Yasui, Edward Ishak
public class ASTOp
{
    public static final int PLUS = 1;
    public static final int MINUS = 2;
    public static final int MULTIPLY = 3;
    public static final int DIVIDE = 4;
    public static final int MOD = 5;
    public static final int AND = 6;
    public static final int OR = 7;
    public static final int PLUSEQUALS = 8;
    public static final int MINUSEQUALS = 9;
    public static final int MULTIPLYEQUALS = 10;
    public static final int DIVIDEEQUALS = 11;
    public static final int MODEQUALS = 12;
    public static final int EQUALS = 13;
    public static final int EQUALEQUALS = 14;
    public static final int NOTEQUALS = 15;
    public static final int GREATERTHAN = 16;
    public static final int LESSTHAN = 17;
    public static final int GREATEREQUALTHAN = 18;
    public static final int LESSEQUALTHAN = 19;
    public static final int PLUSPLUS = 20;
    public static final int MINUSMINUS = 21;
    public static final int DOT = 22;

    public static String getStringValue(int type)
    {
        String str="";

        switch (type)
        {
        case 1:
            str += "+";
            break;
        case 2:
            str += "-";
            break;
        case 3:
            str += "*";
            break;
        case 4:
            str += "/";
            break;
        case 5:
            str += "%";
            break;
        case 6:
            str += "&&";
            break;
        case 7:
            str += "||";
            break;
        case 8:
            str += "+=";
            break;
        case 9:
            str += "-=";
            break;
        case 10:
            str += "*=";
            break;
        case 11:
            str += "/=";
            break;
        case 12:
            str += "%=";
            break;
        case 13:
            str += "=";
```

```
            break;
        case 14:
            str += "==";
            break;
        case 15:
            str += "!=";
            break;
        case 16:
            str += ">";
            break;
        case 17:
            str += "<";
            break;
        case 18:
            str += ">=";
            break;
        case 19:
            str += "<=";
            break;
        case 20:
            str += "++";
            break;
        case 21:
            str += "--";
            break;
        case 22:
            str += ".";
            break;
        }

        return str;
    }

}
```

```
// ASTScope.java
// Authors: Makiko Yasui, Edward Ishak
import java.util.ArrayList;

public class ASTScope extends ArrayList
{
    protected ASTScope parent;
    protected int level;

    public ASTScope(ASTScope p, int l)
    {
        super();
        level = l;
        parent = p;
    }

    public ASTScope getParent()
    {
        return parent;
    }

    public int getLevel()
    {
        return level;
    }

    public Object withinScope(String id)
    {
        Object foundIt = null;
        for (int i=0; i < this.size(); i++)
        {
            if (this.get(i).getClass().getName().equals("ASTFunction"))
            {
                ASTFunction func = ((ASTFunction)this.get(i));
                if (func.getName().equals(id))
                    foundIt = new ASTFunction(func);
            }
            else if (this.get(i).getClass().getName().equals("ASTVariable"))
            {
                ASTVariable var = ((ASTVariable)this.get(i));
                if (var.getName().equals(id))
                    foundIt = var;
            }
            else if (this.get(i).getClass().getName().equals("ASTMyArray"))
            {
                ASTMyArray array = ((ASTMyArray)this.get(i));
                if (array.getName().equals(id))
                    foundIt = array;
            }
        }

        if (foundIt == null && parent != null)
            return parent.withinScope(id);
        else
            return foundIt;
    }

    public boolean withinThisScope(String id)
    {
        for (int i=0; i < this.size(); i++)
        {
            if (this.get(i).getClass().getName().equals("ASTFunction"))
            {
                ASTFunction func = ((ASTFunction)this.get(i));
                if (func.getName().equals(id))
                    return true;
            }
            else if (this.get(i).getClass().getName().equals("ASTVariable"))
            {
                ASTVariable var = ((ASTVariable)this.get(i));
                if (var.getName().equals(id))
```

```
                return true;
            }
            else if (this.get(i).getClass().getName().equals("ASTMyArray"))
            {
                ASTMyArray array = ((ASTMyArray)this.get(i));
                if (array.getName().equals(id))
                    return true;
            }
        }
        return false;
    }

    public ArrayList withinScopeFunc(String id, ArrayList functions)
    {
        for (int i=0; i < this.size(); i++)
        {
            if (this.get(i).getClass().getName().equals("ASTFunction"))
            {
                ASTFunction func = ((ASTFunction)this.get(i));
                if (func.getName().equals(id))
                    functions.add(new ASTFunction(func));
            }
        }
        if (parent != null)
            return parent.withinScopeFunc(id, functions);
        else
            return functions;
    }


    public String toString()
    {
        String str = "";

        str = "level = " + level + "\n";

        for (int i = 0; i < this.size() ; i++)
        {
            if (this.get(i).getClass().getName().equals("ASTFunction"))
                str += ((ASTFunction)this.get(i)).toString() + "\n";
            else if (this.get(i).getClass().getName().equals("ASTVariable"))
                str += ((ASTVariable)this.get(i)).toString() + "\n";
            else if (this.get(i).getClass().getName().equals("ASTMyArray"))
                str += ((ASTMyArray)this.get(i)).toString() + "\n";
            else if (this.get(i).getClass().getName().equals("ASTScope"))
                str += "\t" + ((ASTScope)this.get(i)).toString() + "\n";
        }

        return str;
    }
}
```

```
// ASTType.java
// Authors: Makiko Yasui, Edward Ishak
public class ASTType
{
    public static final int VOID = 1;
    public static final int INT = 2;
    public static final int DOUBLE = 3;
    public static final int STRING = 4;
    public static final int BOOLEAN = 5;
    public static final int IMAGE = 6;
    public static final int COLOR = 7;
    public static final int POINT = 8;
    public static final int PIXEL = 9;
    public static final int BLOB = 10;
    public static final int RECTANGLE = 11;
    public static final int OVAL = 12;

    public static int getTypeValue(String t)
    {
        if (t.equals("void"))
            return ASTType.VOID;
        else if (t.equals("int"))
            return ASTType.INT;
        else if (t.equals("double"))
            return ASTType.DOUBLE;
        else if (t.equals("string"))
            return ASTType.STRING;
        else if (t.equals("boolean"))
            return ASTType.BOOLEAN;
        else if (t.equals("Color"))
            return ASTType.COLOR;
        else if (t.equals("Point"))
            return ASTType.POINT;
        else if (t.equals("Image"))
            return ASTType.IMAGE;
        else if (t.equals("Pixel"))
            return ASTType.PIXEL;
        else if (t.equals("Blob"))
            return ASTType.BLOB;
        else if (t.equals("Rectangle"))
            return ASTType.RECTANGLE;
        else if (t.equals("Oval"))
            return ASTType.OVAL;
        return ASTType.VOID;
    }

    public static String getStringValue(int type)
    {
        String str="";

        switch (type)
        {
        case 1:
            str += "void";
            break;
        case 2:
            str += "int";
            break;
        case 3:
            str += "double";
            break;
        case 4:
            str += "string";
            break;
        case 5:
            str += "boolean";
            break;
        case 6:
            str += "Image";
            break;
        case 7:
```

```
            str += "Color";
            break;
        case 8:
            str += "Point";
            break;
        case 9:
            str += "Pixel";
            break;
        case 10:
            str += "Blob";
            break;
        case 11:
            str += "Rectangle";
            break;
        case 12:
            str += "Oval";
            break;
        }

        return str;
    }

    public static int checkSubType(int type, String subtype)
    {
        if (type == INT || type == DOUBLE || type == STRING || type == BOOLEAN || type ==
VOID)
            return -1;

        if (type == IMAGE)
        {
            if (subtype.equals("width") || subtype.equals("height"))
                return INT;
            else if (subtype.equals("filename"))
                return STRING;
        }

        if (type == COLOR)
        {
            if (subtype.equals("r") || subtype.equals("g") || subtype.equals("b") ||
subtype.equals("a"))
                return INT;
        }

        if (type == POINT)
        {
            if (subtype.equals("x") || subtype.equals("y"))
                return INT;
        }

        if (type == PIXEL)
        {
            if (subtype.equals("point"))
                return POINT;
            else if (subtype.equals("color"))
                return COLOR;
        }

        if (type == RECTANGLE || type == OVAL)
        {
            if (subtype.equals("x") || subtype.equals("y") || subtype.equals("width") ||
subtype.equals("height"))
                return INT;
        }

        return -1;
    }
}
```

```
// ASTVariable.java
// Authors: Makiko Yasui, Edward Ishak
public class ASTVariable
{
    protected int type;
    protected String name;
    protected Object value;
    protected boolean isConstant;
    protected int arrayDim;

    public ASTVariable(String t, String n, Object v, boolean b)
    {
        type = ASTType.getTypeValue(t);
        name = n;
        value = v;
        isConstant = b;
        arrayDim = 0;
    }

    public ASTVariable(String t, String n, Object v, boolean b, int a)
    {
        type = ASTType.getTypeValue(t);
        name = n;
        value = v;
        isConstant = b;
        arrayDim = a;
    }

    public int getType()
    {
        return type;
    }

    public String getName()
    {
        return name;
    }

    public Object getValue()
    {
        return value;
    }

    public boolean isConst()
    {
        return isConstant;
    }

    public int getArrayDim()
    {
        return arrayDim;
    }

    public void setValue(Object v)
    {
        value = v;
    }

    public String toString()
    {
        String str = "variable: type = " + ASTType.getStringValue(type);

        str += ", name = " + name;
        if (value != null)
            str += ", val = " + value.toString();
        else
            str += ", val = null";
        str += ", const = " + isConstant;
        return str;
    }
}
```

```
// CILYColor.java
// Authors: Makiko Yasui
public class CILYColor
{
    public int r;
    public int g;
    public int b;

    public CILYColor(int r_, int g_, int b_)
    {
        r = r_;
        g = g_;
        b = b_;
    }
}
```

```
// CILYCompiler.java
// Authors: Edward Ishak
import java.io.*;

class CILYCompiler
{
    public static void main(String[] args)
    {
        CILYCompiler compiler = new CILYCompiler();
        compiler.compile(args[0], null, false);
    }

    public String compile(String fileName, PrintWriter out, boolean verbose)
    {
        if (verbose)
        {
            System.out.print("Compiling " + fileName + " ... ");
        }
         try
        {
//            CILYLexer lexer = new CILYLexer(new DataInputStream(System.in));
             CILYLexer lexer = new CILYLexer(new DataInputStream(new FileInputStream(new
File(fileName))));
            CILYLexer.lineNumber = 1;
             CILYParser parser = new CILYParser(lexer);
            CILYParser.fileOut = out;
            CILYParser.foundSyntaxError = false;
             parser.program();
            if (!CILYParser.foundSyntaxError)
            {
                CILYTreeParser.filename = fileName.substring(0, fileName.indexOf("."));
                CILYTreeParser treeParser = new CILYTreeParser();
                treeParser.program(parser.getAST());
            }
            //treeParser.globaldeclaration(parser.getAST().getFirstChild());
         }
        catch(Exception e)
        {
             System.err.println("exception: " + e);
         }

        if (verbose)
        {
            System.out.println(" done!");
        }

        return null;
    }
}
```

```
// CILYError.java
// Authors: Makiko Yasui, Edward Ishak
import java.io.*;

public class CILYError
{
    public CILYError()
    {
    }

    public static boolean printError(String s, int line)
    {
        if (CILYParser.fileOut != null)
            CILYParser.fileOut.println("Error at line " + line + ": " + s);
        else
            System.err.println("Error at line " + line + ": " + s);
        return true;
    }
}
```

```
// CILYImage.java
// Authors: Makiko Yasui
import java.awt.*;
import java.awt.image.*;

public class CILYImage
{
    public String filename;
    public BufferedImage bi;
    public int width;
    public int height;

    public CILYImage()
    {
        filename = "defaultImage.jpg";
        bi = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
        width = bi.getWidth(null);
        height = bi.getHeight(null);
    }

    public CILYImage(String filename_)
    {
        filename = filename_;
        bi = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
    }
}
```

```
// CILYLibrary.java
// Authors: Jiwan Choi, Jonathon Liu, Makiko Yasui, Edward Ishak
import java.awt.*;
import java.awt.image.*;
import java.awt.geom.AffineTransform;
import java.io.*;
import java.util.*;
import com.sun.image.codec.jpeg.*;

public class CILYLibrary
{

    public static final int CW_90 = 1;
    public static final int CW_180 = 2;
    public static final int CW_270 = 3;

    public static CILYImage Open(String name)
    {
        try
        {
            //load image
            CILYImage image = new CILYImage(name);
            Image i = Toolkit.getDefaultToolkit().getImage(name);
            MediaTracker mediaTracker = new MediaTracker(new Frame());
            mediaTracker.addImage(i, 0);
            mediaTracker.waitForID(0);
            int iw = i.getWidth(null);
            int ih = i.getHeight(null);

            //convert to bufferedImage from image
            image.bi = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
            Graphics2D big = image.bi.createGraphics();
            big.drawImage(i,0,0,null);
            return image;
        }
        catch (Exception e)
        {
            System.err.println("Error: unable to open image file");
            return null;
        }
    }

    public static void Blur(CILYImage image)
    {
        float[] elements = { .1111f, .1111f, .1111f,
                             .1111f, .1111f, .1111f,
                             .1111f, .1111f, .1111f};

        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        AffineTransform at = new AffineTransform();
        at.scale(1.0, 1.0);
        BufferedImageOp biop = null;

        BufferedImage bimg = new BufferedImage(iw,ih,BufferedImage.TYPE_INT_RGB);

        Kernel kernel = new Kernel(3, 3, elements);
        ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
        cop.filter(image.bi,bimg);
        biop = new AffineTransformOp(at, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);

        Graphics2D g2 = bimg.createGraphics();
        g2.drawImage(bimg, biop, 0,0);
        image.bi = bimg;
    }

    public static void Sharpen(CILYImage image)
    {
        float[] elements = { 0.0f, -1.0f, 0.0f,
                             -1.0f,  5.f, -1.0f,
```

```
                        0.0f, -1.0f,  0.0f};

    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    AffineTransform at = new AffineTransform();
    at.scale(1.0, 1.0);
    BufferedImageOp biop = null;

    BufferedImage bimg = new BufferedImage(iw,ih,BufferedImage.TYPE_INT_RGB);

    Kernel kernel = new Kernel(3, 3, elements);
    ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
    cop.filter(image.bi,bimg);
    biop = new AffineTransformOp(at, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);

    Graphics2D g2 = bimg.createGraphics();
    g2.drawImage(bimg, biop, 0,0);
    image.bi = bimg;
}

public static void DetectEdges(CILYImage image)
{
    float[] elements = { 0.0f, -1.0f, 0.0f,
                         -1.0f, 4.f, -1.0f,
                         0.0f, -1.0f, 0.0f};

    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    AffineTransform at = new AffineTransform();
    at.scale(1.0, 1.0);
    BufferedImageOp biop = null;

    BufferedImage bimg = new BufferedImage(iw,ih,BufferedImage.TYPE_INT_RGB);

    Kernel kernel = new Kernel(3, 3, elements);
    ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
    cop.filter(image.bi,bimg);
    biop = new AffineTransformOp(at, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);

    Graphics2D g2 = bimg.createGraphics();
    g2.drawImage(bimg, biop, 0,0);
    image.bi = bimg;
}


public static void Brighten(CILYImage image, double scale)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    AffineTransform at = new AffineTransform();
    at.scale(1.0, 1.0);
    BufferedImageOp biop = null;

    BufferedImage bimg = new BufferedImage(iw,ih,BufferedImage.TYPE_INT_RGB);

    RescaleOp rop = new RescaleOp((float)scale, (float)scale, null);
    rop.filter(image.bi,bimg);
    biop = new AffineTransformOp(at, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);

    Graphics2D g2 = bimg.createGraphics();
    g2.drawImage(bimg, biop, 0,0);
    image.bi = bimg;
}

public static void Invert(CILYImage image)
{
    int iw = image.bi.getWidth(null);
```

```
    int ih = image.bi.getHeight(null);

    Raster inRaster = image.bi.getRaster();

    BufferedImage bimg = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
     WritableRaster outRaster = bimg.getRaster();

     int[] color = new int[3];
     int[] newColor = new int[3];

     for (int i = 0; i < iw; i++)
     {
         for (int j = 0; j < ih; j++)
         {
             color = inRaster.getPixel(i, j, color);
             newColor[0] = 255-color[0];
             newColor[1] = 255-color[1];
             newColor[2] = 255-color[2];
             outRaster.setPixel(i, j, newColor);
         }
     }

    image.bi = bimg;
}

public static CILYPixel GetPixelAt(CILYImage image, int x, int y)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    Raster inRaster = image.bi.getRaster();

    BufferedImage bimg = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
     WritableRaster outRaster = bimg.getRaster();

     int[] color = new int[3];

    if (x < iw && x >=0 && y < ih && y >= 0)
    {
         color = inRaster.getPixel(x, y, color);
        return new CILYPixel(color[0], color[1], color[2], x, y);
    }
    else
    {
        return new CILYPixel(0,0,0,0,0);
    }
}

public static CILYColor GetColorAt(CILYImage image, int x, int y)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    Raster inRaster = image.bi.getRaster();

     int[] color = new int[3];

    if (x < iw && x >=0 && y < ih && y >= 0)
    {
         color = inRaster.getPixel(x, y, color);
        return new CILYColor(color[0], color[1], color[2]);
    }
    else
    {
        return new CILYColor(0,0,0);
    }
}


public static void Save(CILYImage image)
{
```

```
    try
    {
        BufferedOutputStream out = new BufferedOutputStream(new
                                   FileOutputStream(image.filename));
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
        JPEGEncodeParam param = encoder.
        getDefaultJPEGEncodeParam(image.bi);
        int quality = 100;
        param.setQuality((float)quality / 100.0f, false);
        encoder.setJPEGEncodeParam(param);
        encoder.encode(image.bi);
    }
    catch(Exception e)
    {
        System.err.println("Error: unable to write to image file");

    }
}

public static void SaveAs(CILYImage image, String filename)
{
    try
    {
        BufferedOutputStream out = new BufferedOutputStream(new
                                   FileOutputStream(filename));
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
        JPEGEncodeParam param = encoder.
        getDefaultJPEGEncodeParam(image.bi);
        int quality = 100;
        param.setQuality((float)quality / 100.0f, false);
        encoder.setJPEGEncodeParam(param);
        encoder.encode(image.bi);
    }
    catch(Exception e)
    {
        System.err.println("Error: unable to write to image file");
    }
}

public static void Copy(CILYImage dstImage, CILYImage srcImage)
{
    int iw = srcImage.bi.getWidth(null);
    int ih = srcImage.bi.getHeight(null);

    Raster inRaster = srcImage.bi.getRaster();

    dstImage.bi = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
     WritableRaster outRaster = dstImage.bi.getRaster();

     int[] color = new int[3];

     for (int i = 0; i < iw; i++)
    {
        for (int j = 0; j < ih; j++)
        {
            outRaster.setPixel(i, j, inRaster.getPixel(i, j, color));
        }
     }
}

public static void Paste(CILYImage dstImage, CILYImage srcImage, int x, int y)
{
    int iw = srcImage.bi.getWidth(null);
    int ih = srcImage.bi.getHeight(null);

    Raster inRaster = srcImage.bi.getRaster();
     WritableRaster outRaster = dstImage.bi.getRaster();

     int[] color = new int[3];

     for (int i = 0; i < iw; i++)
```

```
        {
            for (int j = 0; j < ih; j++)
            {
                if (x+i >= 0 && x+i < dstImage.bi.getWidth() && y+j >= 0 && y+j <
dstImage.bi.getHeight())
                    outRaster.setPixel(x+i, y+j, inRaster.getPixel(i, j, color));
            }
        }
    }

    public static void Paste(CILYImage img, CILYPixel[] pixels)
    {
        int iw = img.bi.getWidth(null);
        int ih = img.bi.getHeight(null);

        WritableRaster outRaster = img.bi.getRaster();

        int[] color = new int[3];

        for (int i = 0; i < pixels.length; i++)
        {
            if (pixels[i].point.x >= 0 && pixels[i].point.x < img.bi.getWidth() &&
pixels[i].point.y >= 0 && pixels[i].point.y < img.bi.getHeight())
            {
                color[0] = pixels[i].color.r;
                color[1] = pixels[i].color.g;
                color[2] = pixels[i].color.b;
                outRaster.setPixel(pixels[i].point.x, pixels[i].point.y, color);
            }
        }
    }

    public static void PasteTransparent(CILYImage dstImage, CILYImage srcImage, int x, int
y, int a)
    {
        if (a < 0)
        {
            a = 0;
        } else if (a > 255)
        {
            a = 255;
        }
        int iw = srcImage.bi.getWidth(null);
        int ih = srcImage.bi.getHeight(null);

        Raster inRaster = srcImage.bi.getRaster();
        Raster inRaster2 = dstImage.bi.getRaster();
        WritableRaster outRaster = dstImage.bi.getRaster();

        int[] color = new int[3];
        int[] color2 = new int[3];

        for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if (x+i >= 0 && x+i < dstImage.bi.getWidth() && y+j >= 0 && y+j <
dstImage.bi.getHeight()) {
                    color = inRaster.getPixel(i,j,color);
                    color2 = inRaster2.getPixel(x+i, y+j, color2);
                    color[0] = (int)(((a/255.0)*color[0] + ((255.0-
a)/255.0)*color2[0])/2.0);
                    color[1] = (int)(((a/255.0)*color[1] + ((255.0-
a)/255.0)*color2[1])/2.0);
                    color[2] = (int)(((a/255.0)*color[2] + ((255.0-
a)/255.0)*color2[2])/2.0);
                    outRaster.setPixel(x+i, y+j, color);
                }
            }
        }
    }
```

```
    public static void PasteTransparent(CILYImage dstImage, CILYPixel[] pixels, int a)
    {
        if (a < 0)
        {
            a = 0;
        } else if (a > 255)
        {
            a = 255;
        }
        int iw = dstImage.bi.getWidth(null);
        int ih = dstImage.bi.getHeight(null);

        Raster inRaster = dstImage.bi.getRaster();
         WritableRaster outRaster = dstImage.bi.getRaster();

         int[] color = new int[3];

         for (int i = 0; i < pixels.length; i++)
        {
            if (pixels[i].point.x >= 0 && pixels[i].point.x < dstImage.bi.getWidth() &&
pixels[i].point.y >= 0 && pixels[i].point.y < dstImage.bi.getHeight())
            {
                color = inRaster.getPixel(pixels[i].point.x, pixels[i].point.y, color);
                color[0] = (int)(((a/255.0)*pixels[i].color.r + ((255.0-
a)/255.0)*color[0])/2.0);
                color[1] = (int)(((a/255.0)*pixels[i].color.g + ((255.0-
a)/255.0)*color[1])/2.0);
                color[2] = (int)(((a/255.0)*pixels[i].color.b + ((255.0-
a)/255.0)*color[2])/2.0);
                outRaster.setPixel(pixels[i].point.x, pixels[i].point.y, color);
            }
         }
    }

    public static CILYPixel[] SelectInverse(CILYImage image, CILYPixel[] pixels)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        Raster inRaster = image.bi.getRaster();

         int[] currentColor = new int[3];

        int[][] p = new int[iw][ih];
        int count = 0;

        for (int i = 0; i < pixels.length - 1; i++)
        {
            if (p[pixels[i].point.x][pixels[i].point.y] != 1)
            {
                count++;
                p[pixels[i].point.x][pixels[i].point.y] = 1;
            }
        }

        CILYPixel[] pix = new CILYPixel[iw * ih - count];
        count = 0;
         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if (p[i][j] == 0)
                {
                    currentColor = inRaster.getPixel(i, j, currentColor);
                    pix[count] = new CILYPixel(currentColor[0], currentColor[1],
currentColor[2], i , j);
                    count++;
                }
            }
         }
```

```
        return pix;
    }

    public static CILYPixel[] SelectRect(CILYImage image, CILYRectangle rect)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        Raster inRaster = image.bi.getRaster();

         int[] currentColor = new int[3];

        int[][] pixels = new int[iw][ih];
        int count = 0;

         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if (i >= rect.x && i < rect.x + rect.width && j >= rect.y && j < rect.y +
rect.height)
                {
                    count++;
                    pixels[i][j] = 1;
                }
            }
         }

        CILYPixel[] pix = new CILYPixel[count];
        count = 0;
         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if (pixels[i][j] == 1)
                {
                    currentColor = inRaster.getPixel(i, j, currentColor);
                    pix[count] = new CILYPixel(currentColor[0], currentColor[1],
currentColor[2], i , j);
                    count++;
                }
            }
         }
        return pix;
    }

    public static CILYPixel[] SelectOval(CILYImage image, CILYOval oval)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        Raster inRaster = image.bi.getRaster();

         int[] currentColor = new int[3];

        int[][] pixels = new int[iw][ih];
        int count = 0;

        int widthSq = oval.width * oval.width;
        int heightSq = oval.height * oval.height;

         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if ((double)((i - oval.x) * (i - oval.x)) / (double)widthSq + (double)((j
- oval.y) * (j - oval.y)) / (double)heightSq <= 1 && i >= 0 && i < iw && j >= 0 && j <
ih)
                {
                    count++;
                    pixels[i][j] = 1;
```

```
                }
            }
        }

        CILYPixel[] pix = new CILYPixel[count];
        count = 0;
        for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if (pixels[i][j] == 1)
                {
                    currentColor = inRaster.getPixel(i, j, currentColor);
                    pix[count] = new CILYPixel(currentColor[0], currentColor[1],
currentColor[2], i , j);
                    count++;
                }
            }
        }
        return pix;
    }

    public static void Crop(CILYImage image, int x, int y, int w, int h)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        if (x < 0)
            x = 0;
        else if (x >= iw)
            x = iw-1;

        if (y < 0)
            y = 0;
        else if (y >= ih)
            y = ih-1;

        if (w < 0)
            w = 0;
        else if (x+w >= iw)
            w = iw - x - 1;

        if (h < 0)
            h = 0;
        else if (y+h >= ih)
            h = ih - y - 1;

        image.bi = image.bi.getSubimage(x, y, w, h);
    }

    public static void Crop(CILYImage image, CILYRectangle rectangle)
    {
        int x = rectangle.x;
        int y = rectangle.y;
        int w = rectangle.width;
        int h = rectangle.height;

        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        if (x < 0)
            x = 0;
        else if (x >= iw)
            x = iw-1;

        if (y < 0)
            y = 0;
        else if (y >= ih)
            y = ih-1;

        if (w < 0)
```

```
            w = 0;
        else if (x+w >= iw)
            w = iw - x - 1;

        if (h < 0)
            h = 0;
        else if (y+h >= ih)
            h = ih - y - 1;

        image.bi = image.bi.getSubimage(x, y, w, h);
    }

    public static void ConvertToGrayScale(CILYImage image)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        Raster inRaster = image.bi.getRaster();

        BufferedImage bimg = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
         WritableRaster outRaster = bimg.getRaster();

         int[] color = new int[3];
         int[] newColor = new int[3];
        int avg;

         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                color = inRaster.getPixel(i, j, color);
                avg = (color[0] + color[1] + color[2])/3;
                newColor[0] = avg;
                newColor[1] = avg;
                newColor[2] = avg;
                outRaster.setPixel(i, j, newColor);
            }
         }

        image.bi = bimg;
    }

    public static CILYPixel[] MagicWand(CILYImage image, CILYColor color, int tolerance)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        Raster inRaster = image.bi.getRaster();

         int[] currentColor = new int[3];

        int[][] pixels = new int[iw][ih];
        int count = 0;

         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                currentColor = inRaster.getPixel(i, j, currentColor);
                if (currentColor[0] > color.r - tolerance && currentColor[0] < color.r +
tolerance && currentColor[1] > color.g - tolerance && currentColor[1] < color.g +
tolerance && currentColor[2] > color.b - tolerance && currentColor[2] < color.b +
tolerance)
                {
                    count++;
                    pixels[i][j] = 1;
                }
            }
         }

        CILYPixel[] pix = new CILYPixel[count];
```

```
        count = 0;
         for (int i = 0; i < iw; i++)
        {
            for (int j = 0; j < ih; j++)
            {
                if (pixels[i][j] == 1)
                {
                    currentColor = inRaster.getPixel(i, j, currentColor);
                    pix[count] = new CILYPixel(currentColor[0], currentColor[1],
currentColor[2], i , j);
                    count++;
                }
            }
         }
        return pix;
    }

    public static void SetColorAt(CILYImage image, CILYColor color, int x, int y)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        int[] c = new int[3];
        c[0] = color.r;
        c[1] = color.g;
        c[2] = color.b;

        WritableRaster outRaster = image.bi.getRaster();

        if (x < iw && x >=0 && y < ih && y >= 0)
            outRaster.setPixel(x, y, c);
    }

    public static void Rotate(CILYImage image, int degrees)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        BufferedImageOp biop = null;
        AffineTransform at = new AffineTransform();
        BufferedImage bimg;

        if (degrees == CW_90)
        {
            at.rotate((Math.toRadians(90)));
            bimg = new BufferedImage(ih, iw,BufferedImage.TYPE_INT_RGB);
        }
        else if (degrees == CW_180)
        {
            at.rotate((Math.toRadians(180)));
            bimg = new BufferedImage(iw, ih,BufferedImage.TYPE_INT_RGB);
        }
        else if (degrees == CW_270)
        {
            at.rotate((Math.toRadians(270)));
            bimg = new BufferedImage(ih, iw,BufferedImage.TYPE_INT_RGB);
        }
        else
        {
            at.rotate((Math.toRadians(0)));
            bimg = new BufferedImage(iw, ih,BufferedImage.TYPE_INT_RGB);
        }

        biop = new AffineTransformOp(at, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);

        Graphics2D g2 = bimg.createGraphics();

        if (degrees == CW_90)
            g2.drawImage(image.bi, biop, ih,0);
        else if (degrees == CW_180)
            g2.drawImage(image.bi, biop, iw,ih);
```

```
    else if (degrees == CW_270)
        g2.drawImage(image.bi, biop, 0,iw);
     else
        g2.drawImage(image.bi, biop, 0,0);

    image.bi = bimg;
}

public static void FlipHorizontal(CILYImage image)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    Raster inRaster = image.bi.getRaster();

    BufferedImage img = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
     WritableRaster outRaster = img.getRaster();

     int[] color = new int[3];

     for (int i = 0; i < iw; i++)
    {
        for (int j = 0; j < ih; j++)
            outRaster.setPixel(iw-1-i, j, inRaster.getPixel(i, j, color));
     }

    image.bi = img;
}

public static void FlipVertical(CILYImage image)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    Raster inRaster = image.bi.getRaster();

    BufferedImage img = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
     WritableRaster outRaster = img.getRaster();

     int[] color = new int[3];

     for (int i = 0; i < iw; i++)
    {
        for (int j = 0; j < ih; j++)
            outRaster.setPixel(i, ih-1-j, inRaster.getPixel(i, j, color));
     }

    image.bi = img;
}

public static void AddBorder(CILYImage image, CILYColor color, int thickness)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    Raster inRaster = image.bi.getRaster();

    BufferedImage bimg = new BufferedImage(iw, ih, BufferedImage.TYPE_INT_RGB);
    WritableRaster outRaster = bimg.getRaster();

    int[] c = new int[3];
    int[] c2 = new int[3];
    c[0] = color.r;
    c[1] = color.g;
    c[2] = color.b;

    for (int i = 0; i < iw; i++)
    {
        for (int j = 0; j < ih; j++)
        {
            c2 = inRaster.getPixel(i, j, c2);
```

```
                if (i < thickness || i > iw-1-thickness || j < thickness || j > ih-1-
thickness)
                    outRaster.setPixel(i, j, c);
                else
                    outRaster.setPixel(i, j, c2);
            }
        }

        image.bi = bimg;
    }

    private static String fromRGBtoHex(CILYColor c)
    {
        return "" + toHex(c.r)+ "" + toHex(c.g) + "" + toHex(c.b) + "";
    }

    private static String toHex(int n)
    {
        if (n == 0)
            return "00";

        n = Math.max(0,n);
        n = Math.min(n,255);
        return "" + "0123456789ABCDEF".charAt((n-n%16)/16) + "" +
"0123456789ABCDEF".charAt(n%16) + "";
    }

    public static void CreateAlbum(String filename,
                        String title,
                        String subtitle,
                        CILYColor bgcolor,
                        CILYColor fgcolor,
                        int style,
                        CILYImage[] imgs)
    {
        if (style == 1)
            CreateAlbum(filename, title, subtitle, bgcolor, fgcolor, 5, 100, 100, imgs);
        else if(style == 2)
            CreateAlbum(filename, title, subtitle, bgcolor, fgcolor, 10, 50, 50, imgs);
    }

    public static void CreateAlbum(String filename,
                                String title,
                                String subtitle,
                                CILYColor bgcolor,
                                CILYColor fgcolor,
                                int numPics,
                                int thumbheight,
                                int thumbwidth,
                                CILYImage[] imgs)
    {

        try
        {
            String dir = GetDir(filename);
            String thumbDir = dir + "thumbnails\\";
            String imageDir = dir + "images\\";

            //create folder for thumbnails and images
            File f = new File(thumbDir);
            f.mkdir();
            f = new File(imageDir);
            f.mkdir();

            //Copy pictures in images folder
            for (int i= 0; i< imgs.length; i++)
            {
                SaveAs(imgs[i], imageDir + GetRelPath(imgs[i].filename));
            }

            for(int s =0; s < imgs.length; s++)
```

```
            {
                CreateThumbnail( imgs[s], dir, thumbwidth, thumbheight);
            }

            //start printing to the html file.
            PrintWriter out = new PrintWriter(new FileWriter(filename));

            //title
            out.println("<html>\n<head>\n<title>" + title + "</title>\n");
            out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=iso-8859-1\"></head>\n");
            //body begins
            out.println("<body bgcolor=\"#" + fromRGBtoHex(bgcolor) + "\">\n<center><span
class=\"title\" style=\"font-family:verdana; font-size:20pt; color:#" +
fromRGBtoHex(fgcolor) + "; text-decoration: none;\">" + title+ "</span><br><span
class=\"subtitle\" style=\"font-family:verdana; font-size:15pt; color:#" +
fromRGBtoHex(fgcolor) + "; text-decoration: none;\">" + subtitle + "</span><br><br>");

            //start table
            out.println("<table width=\""+ (thumbwidth+10)*numPics +"\" border=\"1\"
cellpadding=\"5\" cellspacing=\"0\" bordercolor=\"#" + fromRGBtoHex(fgcolor) +"\">\n");

            int totalPix = imgs.length;
            int totalBoxes = 0;
            if((totalPix%numPics) != 0)
                totalBoxes = totalPix + (numPics-(totalPix%numPics));
            else
                totalBoxes = totalPix;

            //rows
            for(int i=0; i<(totalBoxes/numPics); i++)
            {
                out.println("<tr>\n");

                //columns
                for(int j=0;j<numPics; j++)
                {
                    if (i*numPics+j > (totalPix-1))
                        out.println("<td width=\""+ (thumbwidth+10) +"\" height=\""+
(thumbheight+10) +"\"> </td>\n");
                    else
                        out.println("<td width=\""+(thumbwidth+10) +"\"
height=\""+(thumbheight+10)+"\" align=\"center\" valign=\"middle\"><a href=\"" + imageDir
+ GetRelPath(imgs[numPics*i+j].filename) + "\" target=\"_blank\"><img src=\"" +
GetThumbName(dir, GetRelPath(imgs[numPics*i+j].filename)) + "\"
border=\"0\"></a></td>\n");
                }

                out.println("</tr>\n");
            }
            out.println("</table>\n</center>\n</body>\n</html>");
            out.close();
        }
        catch (Exception e)
        {
            System.out.println("Error: unable to create folders");
        }
    }

    public static void CreateThumbnail(CILYImage image, String dir, int newHeight, int
newWidth)
    {
        int iw = image.bi.getWidth(null);
        int ih = image.bi.getHeight(null);

        double thumbRatio = (double)newWidth / (double)newHeight;
        double imageRatio = (double)iw / (double)ih;

        if (thumbRatio < imageRatio)
            newHeight = (int) (newWidth / imageRatio);
        else
```

```
        newWidth = (int)(newHeight * imageRatio);

    BufferedImage thumbImage = new BufferedImage(newWidth,
                                newHeight, BufferedImage.TYPE_INT_RGB);
    Graphics2D graphics2D = thumbImage.createGraphics();
    graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
    graphics2D.drawImage(image.bi, 0, 0, newWidth, newHeight, null);

    String newName = GetThumbName(dir, GetRelPath(image.filename));

    //save image
    try
    {
        BufferedOutputStream out = new BufferedOutputStream(new
                                    FileOutputStream(newName));
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
        JPEGEncodeParam param = encoder.
        getDefaultJPEGEncodeParam(thumbImage);
        int quality = 100;
        param.setQuality((float)quality / 100.0f, false);
        encoder.setJPEGEncodeParam(param);
        encoder.encode(thumbImage);
    }
    catch(Exception e)
    {
        System.err.println("Error: unable to open file");
    }
}

public static void Resize(CILYImage image, int newWidth, int newHeight)
{
    int iw = image.bi.getWidth(null);
    int ih = image.bi.getHeight(null);

    double thumbRatio = (double)newWidth / (double)newHeight;
    double imageRatio = (double)iw / (double)ih;

    BufferedImage newImage = new BufferedImage(newWidth,
                                newHeight, BufferedImage.TYPE_INT_RGB);
    Graphics2D graphics2D = newImage.createGraphics();
    graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
    graphics2D.drawImage(image.bi, 0, 0, newWidth, newHeight, null);

    image.bi = newImage;
}

private static String GetRelPath(String path)
{
    String dir = "";
    int  j = path.lastIndexOf('\\');
    if (j >= 0){
        dir = path.substring(j+1, path.length());
        return dir;
    }
    else
        return path;
}

private static String GetDir(String path)
{
    String dir = "";
    int  j = path.lastIndexOf('\\');
    if (j >= 0)
        dir = path.substring(0, j+1);
    return dir;
}

private static String GetThumbName(String dir, String name)
{
```

```
        String thumbName = name.substring(0, name.length()-4);
        thumbName = dir + "thumbnails\\" + thumbName + "_t.jpg";
        return thumbName;
    }

    public static CILYImage[] OpenDir(String path)
    {
        try
        {
            File fl = new File(path);
            File[] fileArray = fl.listFiles();
            Vector fileVector = new Vector();
            for ( int h=0; h<fileArray.length; h++)
            {
                fileVector.add(fileArray[h]);
            }

            //checking for jpgs
            for ( int j = 0; j < fileVector.size(); j++)
            {
                String ext = ((File)fileVector.elementAt(j)).getName();
                String substr = ext.substring( (ext.length() - 4), ext.length() );
                if( ! substr.equalsIgnoreCase(".jpg") )
                {
                    fileVector.remove(j);
                    j--;
                }
            }

            CILYImage[] imgArray = new CILYImage[fileVector.size()];

            for ( int i = 0; i< fileVector.size(); i++)
            {
                imgArray[i] = Open(((File)fileVector.get(i)).getAbsolutePath() );
            }
            return imgArray;

        }
        catch (Exception e)
        {
            System.out.println("Error: unable to open directory");
            return null;
        }
    }
}
```

```
// CILYOval.java
// Authors: Makiko Yasui
public class CILYOval
{
    public int x;
    public int y;
    public int width;
    public int height;

    public CILYOval(int x_, int y_, int width_, int height_)
    {
        x = x_;
        y = y_;
        width = width_;
        height = height_;
    }
}
```

```
// CILYPixel.java
// Author: Makiko Yasui
public class CILYPixel
{
    public CILYColor color;
    public CILYPoint point;

    public CILYPixel(int r, int g, int b, int x, int y)
    {
        color = new CILYColor(r, g, b);
        point = new CILYPoint(x, y);
    }
}
```

```
// CILYPoint.java
// Author: Makiko Yasui
public class CILYPoint
{
    public int x;
    public int y;

    public CILYPoint(int x_, int y_)
    {
        x = x_;
        y = y_;
    }
}
```

```
// CILYRectangle.java
// Author: Makiko Yasui
public class CILYRectangle
{
    public int x;
    public int y;
    public int width;
    public int height;

    public CILYRectangle(int x_, int y_, int width_, int height_)
    {
        x = x_;
        y = y_;
        width = width_;
        height = height_;
    }
}
```

```
// CILYTestCases.java
// Author: Edward Ishak
import java.io.*;

public class CILYTestCases {
    public static void main(String args[]) {
        CILYCompiler compiler = new CILYCompiler();
        boolean verbose = true, seenError = false;
        String expectedText = "";
        PrintWriter fileOut = null;

        String[] testFiles = {
            "testcases/comments/basic_comments",
            "testcases/comments/advanced_comments",
            "testcases/comments/same_line_comments",
            "testcases/array/size_declared_and_initialized",
            "testcases/array/initialized_by_function",
            "testcases/array/inconsistent_dimensions_01",
            "testcases/array/inconsistent_dimensions_02",
            "testcases/array/array_assignment",
            "testcases/functions/functions_with_no_parameters",
            "testcases/functions/functions_with_parameters",
            "testcases/functions/nested_functions",
            "testcases/functions/return_types",
            "testcases/functions/missing_keyword",
            "testcases/functions/missing_return",
            "testcases/functions/missing_arg_types",
            "testcases/functions/overloading_functions",
            "testcases/functions/returning_array",
            "testcases/functions/array_as_parameter",
            "testcases/functions/constructor",
            "testcases/functions/library_functions",
            "testcases/functions/overloaded_library_functions",
            "testcases/global/basic_global_access",
            "testcases/negative/keyword_as_id",
            "testcases/negative/mismatching_brackets",
            "testcases/negative/mismatching_curly_braces",
            "testcases/negative/mismatching_parens",
            "testcases/negative/missing_comma",
            "testcases/negative/missing_semicolon",
            "testcases/negative/no_return_type",
            "testcases/negative/non_int_array_iterator",
            "testcases/string/break_up_string",
            "testcases/string/nested_quotes",
            "testcases/string/printing_variables",
            "testcases/arrow/arrow_test",
            "testcases/types/primitive_types",
            "testcases/types/mismatching_types",
            "testcases/types/precedence",
            "testcases/types/subtypes",
            "testcases/types/plusplus",
            "testcases/types/plusequals",
            "testcases/loops/foreach_with_images",
            "testcases/loops/while_non_boolean_exp",
            "testcases/operators/allops",
        };

        for (int i=0; i < testFiles.length; i++) {
            try {
                fileOut = new PrintWriter(new FileWriter(testFiles[i] + ".out"));
                compiler.compile(testFiles[i] + ".cily", fileOut, verbose);
                fileOut.close();
                expectedText = getFileContents(testFiles[i] + ".exp");
                if (!expectedText.equals(getFileContents(testFiles[i] + ".out"))) {
                    System.out.println("Test file " + testFiles[i] + ".cily FAILED!");
                    seenError = true;
                }
            }
            catch (IOException e)
            {
                System.out.println("Cannot write to file " + testFiles[i] + ".out");
```

```
            }
        }

        if (seenError) {
            System.out.println("Test cases have errors!");
        } else {
            System.out.println("Test cases passed!");
        }
    }

    public static String getFileContents(String fileName) {
        String line, retVal = "";
        BufferedReader in;
        try {
            in = new BufferedReader(new FileReader(fileName));

            while ((line = in.readLine()) != null) {
                retVal += line;
            }
        }
        catch (IOException e)
        {
            System.err.println("cannot open output file " + fileName + " for test case
comparisons");
        }

        return retVal;
    }

}
```

```
// CodeGenerator.java
// Author: Makiko Yasui, Edward Ishak
import java.io.*;
import java.util.ArrayList;

public class CodeGenerator
{
    private String className = "CILYProgram";
    private PrintWriter fileOut = null;

    public CodeGenerator(String className)
    {
        this.className = getRelPath(className);
        try
        {
            fileOut = new PrintWriter(new FileWriter(this.className + ".java"));
        }
        catch (IOException e)
        {
            System.err.println("ERROR: unable to write to file " + this.className +
".java");
        }
    }

    private String getRelPath(String path)
    {
        String filename = "";
        int  j = path.lastIndexOf('/');
        if (j >= 0){
            filename = path.substring(j+1, path.length());
            return filename;
        }
        else
            return path;
    }

    public int printClassBeg(int numTabs)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("import java.io.*;");
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("public class " + className + " {");
        return numTabs+1;
    }

    public int printClassEnd(int numTabs)
    {
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
        fileOut.close();
        return numTabs-1;
    }

    public int printMainBeg(int numTabs)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("public static void main(String args[]) {");
        for (int i=0; i < numTabs+1; i++)
        {
```

```
            fileOut.print("\t");
        }
        fileOut.println("try {");
        return numTabs+2;
    }

    public int printMainEnd(int numTabs)
    {
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("} catch (ArrayIndexOutOfBoundsException aioobe) {");
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("System.err.println(\"RUNTIME ERROR: Array index out of
bounds\");");
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("} catch (Exception e) {");
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("System.err.println(\"RUNTIME ERROR: Unable to read or write to
file\");");
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
        for (int i=0; i < numTabs-2; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
        return numTabs-2;
    }

    public int printFunctionBeg(int numTabs, int type, int arrayDim, String funcName,
ArrayList args)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.print("public static " + getTypeName(type));
        if (arrayDim == 2)
            fileOut.print("[][] " + funcName);
        else if (arrayDim == 1)
            fileOut.print("[] " + funcName);
        else
            fileOut.print(" " + funcName);

        fileOut.print("(");

        for (int i = 0; i < args.size(); i++)
        {
            if (i > 0)
            {
                fileOut.print(",");
            }
            ASTVariable var = (ASTVariable)(args.get(i));
            fileOut.print(getTypeName(var.getType()) + "");
            if (var.getArrayDim() == 1)
                fileOut.print("[]");
            else if (var.getArrayDim() == 2)
```

```
                fileOut.print("[][]");
            fileOut.print(" " + var.getName());
        }

        fileOut.println(")");
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("{");
        return numTabs+1;
    }

    public int printFunctionEnd(int numTabs)
    {
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
        return numTabs-1;
    }

   public int printVarDeclaration(int numTabs, int type, String varName, String initValue,
boolean isGlobal, boolean isConst)
    {
        //indentations
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }

        if (isGlobal)
        {
            fileOut.print("public static ");
        }

        if (isConst) {
            fileOut.print("final ");
        }

        fileOut.print(getTypeName(type));

        fileOut.print(" " + varName);
        if (initValue != null) {
            fileOut.print(" = " + initValue);
        }
        fileOut.println(";");

        return numTabs;
    }

    public int printArrayDeclaration(int numTabs, int type, String varName, String size1,
String size2, int dimension, String initValue, boolean isGlobal)
    {
        //indentations
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }

        if (isGlobal)
        {
            fileOut.print("public ");
        }

        fileOut.print(getTypeName(type));

        if (size1 == null && size2 == null)
        {
            fileOut.print("[]");
```

```
            if (dimension == 2)
            {
                fileOut.print("[]");
            }
            fileOut.print(" " + varName);
            fileOut.print(" = " + initValue);
        }
        else if (size2 == null)
        {
            fileOut.print("[] " + varName + "= new " + getTypeName(type) + "[" + size1 +
"] ");
        }
        else
        {
            fileOut.print("[][] " + varName + "= new " + getTypeName(type) + "[" + size1 +
"][" + size2 + "] ");
        }

        fileOut.println(";");

        return numTabs;
    }

    public int printArrayDeclaration(int numTabs, int type, String varName, String size1,
String size2, String initValue, int dimension, boolean isGlobal)
    {
        //indentations
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }

        if (isGlobal)
        {
            fileOut.print("public ");
        }

        fileOut.print(getTypeName(type));

        if (size2 == null)
        {
            fileOut.print("[] " + varName + "= new " + getTypeName(type) + "[" + size1 +
"] ");
        }
        else
        {
            fileOut.print("[][] " + varName + "= new " + getTypeName(type) + "[" + size1 +
"][" + size2 + "] ");
        }

        fileOut.println(";");

        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("for (int _ = 0; _ < " + size1 + "; _++)");
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("{");
        for (int i=0; i < numTabs + 1; i++)
        {
            fileOut.print("\t");
        }
        if (size2 == null)
        {
            fileOut.println(varName + "[_] = " + initValue + ";");
        }
        else
```

```
    {
        fileOut.println("for (int __ = 0; __ < " + size2 + "; __++)");
        for (int i=0; i < numTabs + 1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("{");
        for (int i=0; i < numTabs + 2; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println(varName + "[_][__] = " + initValue + ";");
        for (int i=0; i < numTabs + 1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
    }
    for (int i=0; i < numTabs; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println("}");

    return numTabs;
}

public String getTypeName(int type)
{
    String retVal = "";

    switch (type)
    {
    case 1:
        retVal = "void";
        break;
    case 2:
        retVal = "int";
        break;
    case 3:
        retVal = "double";
        break;
    case 4:
        retVal = "String";
        break;
    case 5:
        retVal = "boolean";
        break;
    case 6:
        retVal = "CILYImage";
        break;
    case 7:
        retVal = "CILYColor";
        break;
    case 8:
        retVal = "CILYPoint";
        break;
    case 9:
        retVal = "CILYPixel";
        break;
    case 10:
        retVal = "CILYBlob";
        break;
    case 11:
        retVal = "CILYRectangle";
        break;
    case 12:
        retVal = "CILYTriangle";
        break;
    case 13:
        retVal = "CILYOval";
```

```
            break;
        case 14:
            retVal = "CILYHeart";
            break;
        }

        return retVal;
    }

    public int printFuncDeclBeg(int numTabs, int retType, String funcName)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.print("public " + getTypeName(retType) + " " + funcName + "(");
        return numTabs;
    }

    public int printWhileBeg(int numTabs, String expr)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("while (" + expr + ") {");
        return numTabs+1;
    }

    public int printWhileEnd(int numTabs)
    {
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
        return numTabs-1;
    }

    public int printForeachBeg(int numTabs, String iterator, String arrayName)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("for (int " + iterator + " = 0; " + iterator + " < " + arrayName +
".length; " + iterator + "++) {");
        return numTabs+1;
    }

    public int printForeachEnd(int numTabs)
    {
        for (int i=0; i < numTabs-1; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("}");
        return numTabs-1;
    }

    public int printIfBeg(int numTabs, String expr)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("if (" + expr + ") {");
        return numTabs+1;
    }

    public int printElseIf(int numTabs, String expr)
```

```
{
    for (int i=0; i < numTabs-1; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println("else if (" + expr + ") {");
    return numTabs;
}

public int printElse(int numTabs)
{
    for (int i=0; i < numTabs-1; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println("else {");
    return numTabs;
}

public int printIfEnd(int numTabs)
{
    for (int i=0; i < numTabs-1; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println("}");
    return numTabs-1;
}

public int printBlockEnd(int numTabs)
{
    for (int i=0; i < numTabs-1; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println("}");
    return numTabs;
}

public int printAssignStmt(int numTabs, String expr)
{
    for (int i=0; i < numTabs; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println(expr + ";");
    return numTabs;
}

public int printFunctionCallStmt(int numTabs, String expr)
{
    for (int i=0; i < numTabs; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println(expr + ";");
    return numTabs;
}

public int printReturnStmt(int numTabs, String expr)
{
    for (int i=0; i < numTabs; i++)
    {
        fileOut.print("\t");
    }
    fileOut.println("return " + expr + ";");
    return numTabs;
}

public int printPrintStmt(int numTabs, String s)
{
```

```
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("System.out.print(" + s + ");");
        return numTabs;
    }

    public int printPrintlnStmt(int numTabs, String s)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println("System.out.println(" + s + ");");
        return numTabs;
    }

    public int print(int numTabs, String s)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.print(s);
        return numTabs;
    }

    public int println(int numTabs, String s)
    {
        for (int i=0; i < numTabs; i++)
        {
            fileOut.print("\t");
        }
        fileOut.println(s);
        return numTabs;
    }

    public void generate()
    {
        try
        {
            PrintWriter fOut = new PrintWriter(new FileWriter(className + ".bat"));
            fOut.println("@echo off");
            fOut.println("del " + className + ".class");
            fOut.println("javac " + className + ".java");
            fOut.println("java " + className);
            fOut.close();
        }
        catch (IOException e)
        {
            System.err.println("Couldn't create file " + className + ".bat");
        }
    }

    public static void main(String args[])
    {
        int numTabs=0;
        CodeGenerator codeGen = new CodeGenerator("TestClass");
        numTabs = codeGen.printClassBeg(numTabs);
        numTabs = codeGen.printVarDeclaration(numTabs, ASTType.STRING, "a", "\"(2+3) *
3\"", true, false);
        numTabs = codeGen.printVarDeclaration(numTabs, ASTType.BOOLEAN, "b", "true", true,
true);
        numTabs = codeGen.printMainBeg(numTabs);
        numTabs = codeGen.println(numTabs, "System.out.println(\"hi\");");
        numTabs = codeGen.printWhileBeg(numTabs, "x < 3");
        numTabs = codeGen.printWhileEnd(numTabs);
        numTabs = codeGen.printForeachBeg(numTabs, "i", "arr");
        numTabs = codeGen.printForeachEnd(numTabs);
        numTabs = codeGen.printIfBeg(numTabs, "true");
```

```
        numTabs = codeGen.printElseIf(numTabs, "true");
        numTabs = codeGen.printElseIf(numTabs, "true");
        numTabs = codeGen.printElse(numTabs);
        numTabs = codeGen.printForeachEnd(numTabs);
        numTabs = codeGen.printMainEnd(numTabs);
        numTabs = codeGen.printClassEnd(numTabs);
        codeGen.generate();
    }
}
```