IML
Image Manipulation Language

Travis J. Galoppo
tjg2107@columbia.edu

1. Introduction

IML (Image Manipulation Language) is an interpreted programming language, built on top of the Java environment, for the rapid and easy development of image transformation tools. The language provides easy to use mechanisms for creating, loading, saving, and per-pixel processing of RGB raster images.

1.1 Motivation

Programmatically working with raster graphics can be challenging on many levels; the variety of file formats alone can keep a developer busy for years. Dealing with the complexities of bit masking, width vs. pitch, and colorspace issues can be very confusing. Although a number of libraries exist to help make the task easier, none seem to strike the ideal balance between ease of use and functionality.

Furthermore, while a number of very comprehensive software packages exist for image manipulation (i.e. Photoshop, GIMP), writing custom plug-ins for these packages remains a daunting task; also, batch processing of images with these packages, while possible, is limited.

The goal of IML is to provide a means of creating image transformation tools that can be used to batch process images.

1.2 IML

The crux of the language is the built-in “image” type, which allows for the easy loading, saving, creation, and per-pixel manipulation of RGB raster graphics. Images are broken down into pixels (addressable by (x,y) coordinate), which are further broken down into the individual color channels. The pixel type also has a built-in conversion to a 32-bit integer type, allowing for traditional bit-masking of color components, should the developer have a use for such.

By simplifying the details of accessing the raster pixel data, the programmer can concentrate on the implementation of image transforms.

1.3 Considerations

1.3.1 Special Purpose

IML is a special purpose language; as such, it provides only the constructs essential to the task of image manipulation, and limited, if any, support for general application development.

1.3.2 Simplicity

The special purpose nature of IML means that the language itself is quite small, making it relatively easy to learn. Furthermore, the built-in image and pixel types make the normally tricky task of raster manipulation quite easy.

1.3.3 Portability

Being rooted in the Java language, IML programs are architecture neutral and completely portable. Write once, run everywhere! (Well, ok, everywhere there is a Java runtime installed)

1.4 Sneak Peek

Syntax

IML syntax is a hybrid of Pascal and C++. Variable and function declarations take a Pascal-like form; for example:

To define a function, foo, which takes an integer, a, as an argument and returns an integer, you would use the following declaration:

```
function foo(a : integer) : integer
```

To define a function, bar, which takes an image, img, as an argument and returns nothing, you would use the following declaration:

```
function bar(img : image) : null
```

Variables defined within a function are preceded by the 'var' keyword:

```
var i, j : integer;
```

Most statements, however, are more familiar to C and C++ programmers:

```
for(i=0; i<10; i++){
  if(i % 2){
    j = i * 5;
  } else {
    j = i / 2;
  }
}
```

A function to perform a 50% alpha blend of two pixels would look like this:

```
function alphablend(p1 : pixel, p2 : pixel) : pixel {
  var p3 : pixel;
  p3.red = (p1.red + p2.red) / 2;
  p3.green = (p1.green + p2.green) / 2;
  p3.blue = (p1.blue + p2.blue) / 2;
  return p3;
}
```

As noted earlier, however, the language supports an implied conversion between the pixel type and the integer type, allowing code such as:

```
/* Invert pixel - for each color channel, c, c = 255-c */
function invert(p1 :  pixel) : pixel {
  var p2 : pixel;
  p2 = ~p1 & 0x00FFFFFF;
  return p2;
}
```

Furthermore, while user defined classes are not permitted in IML, the built in image type is a class with both static and instance methods. For example:

To load an image, we can say:

```
var myImage : image;
myImage = image::open("someimage.jpg");
```

To create a new image, we can say:

```
var newImage : image;
newImage = image::create(640, 480);
```

we could then save that image:

```
image::save(newImage, "myimage.jpg");
```

Pixel data within the image is accessed by treating the image object as a two-dimensional array of pixels:

```
var myPixel : pixel;
myPixel = newImage[0,0]; /* x,y pixel coordinates */
```

Hopefully, this brief introduction to IML syntax has sparked your interest!

1.5 Summary

The IML language provides a simple platform for implementing image transform routines, which can subsequently be used for batch processing of image files. Due to its special purpose nature, it is easy to learn and simplifies the task of working with raster graphics.

2. Tutorial

2.1 Getting Started

2.1.1 Requirements

IML is an interpreted language built on top of the Java runtime; consequentially, you will need to have the Java runtime installed on you're computer.

2.1.2 Running IML programs

You invoke the IML interpreter via the command line, ie:

$ java IML myprogram.iml

where **myprogram.iml** is the IML source program you wish to run.

There are two possible options to alter the behavior of the interpreter; the -**o** option invokes the optimizer before executing the program, and the -**v** options turns on "verbose" mode, which will show a dump of the AST before and after optimization (if turned on). Thus, to turn on both optimizing and verbose mode, invoke IML as so:

$ java IML –ov myprogram.iml

2.2 Coding in IML

IML syntax is fashioned after the C and C++ languages, with Pascal style function and variable declarations; this brief tutorial is designed to introduce the IML specific **image** and **pixel** data types, and the operations that can be performed on them. For a complete reference of the IML syntax, please refer to section 3 (Language Reference Manual).

The core of the IML language is the built in **image** and **pixel** types; an instance of the image type is essentially a two dimensional array of pixels, with properties to describe the width and height of the image. The pixel type is an integer compatible value with properties for the red, green, and blue color components. The image type (class) also has three static methods associated with it for loading, saving, and creating new images.

2.3.1 The pixel data type

The pixel data type is effectively a 24 bit unsigned integer; the value is split into three 8 bit values to represent red, green, and blue intensities which combine to form different colors. The color channels can be accessed like so:

```
var p : pixel;
p.red = 128;
p.green = 128;
p.blue = 128;
```

Each color component can range in value from 0 to 255 (2^8-1), and the IML interpreter will give an error if an attempt is made to set a color component to an invalid value.

It is legal to treat a pixel as a normal integer, and all integer operations are valid; note that the internal representation is a 32 bit integer, and the top high order 8 bits will be stripped when the value is used in computation.

2.3.2 The image data type

The image type is the bread and butter of the IML language. An image object is a two dimensional array of pixels, ultimately representing a 2D RGB raster image. Individual pixels can be accessed like so:

```
var p : pixel;
var img : image;

.
.
p = img[0,0]; /* get pixel at image coordinates 0,0 (top left) */
.
.
```

Note the the coordinate order is [x,y], and that the origin is top left.

The image type has two properties, **width** and **height**, accessible like so:

```
.
.
w = img.width;
h = img.height;
```

```
.
.
```

The image type also has three associated static methods: **load**, **save**, and **create**:

```
/* This program produces a negative of image "someimage.jpg" */
program NegateImage:

main(){
	var img1, img2 : image;
	var x, y : integer;
	var p : pixel;

	img1 = image::load("someimage.jpg");
	img2 = image::create(img1.width, img1.height);

	for(y=0; y<img1.height; y=y+1){
		for(x=0; x<img1.width; x=x+1){
			p = img1[x,y];
			p.red = 255-p.red;
			p.green = 255-p.green;
			p.blue = 255-p.blue;
			img2[x,y] = p;
		}
	}

	image::save(img2,"negimage.jpg");
}
end.
```

3. Language Reference Manual

3.1 Introduction

IML is a computer language, based loosely on the C and Pascal languages, designed for easy manipulation of RGB raster images. IML is an interpreted language, built on top of the Java platform, and is therefore platform independent.

At the time of this writing, absolutely no software is written in IML, as the interpreter is being developed in parallel with this reference manual. As such, the final language is subject to vary slightly from this version of the manual.

3.2 Lexical conventions

IML distinguishes between five kinds of tokens: identifiers, keywords, numeric constants, expression operators, and separators. Blanks, tabs, newlines, and comments (as to be described) are ignored, other than to separate tokens.

3.2.1 Comments

IML employs C style comments, starting with the characters /* and terminating with the characters */.

3.2.2 Identifiers

An identifier is a sequence of letters, digits, and underscores; identifiers must start with a letter (underscore is not allowed), and can be any length. Upper and lower case letters are considered different.

3.2.3 Keywords

The following identifiers are reserved as keywords, and are not valid user defined identifiers:

integer	if
real	else
string	for
null	while
return	program
end	do
image	pixel
var	function
main	print

3.2.4 Constants

There are both integer and real (floating point) constants:

3.2.4.1 Integer Constants

An integer constant is a sequence of digits. Integer constants are always interpreted as base 10. Future versions of the language will include hexadecimal notation, similar to that of C.

3.2.4.2 Real Constants

Real, or floating point, constants consist of a series of digits followed by a . followed, possibly, by more digits. For example: 1.24 1234.5 1. are all Real numbers. Note that a number of the form .25 is not. Also, there is no provision for C style "scientific notation".

3.2.5 Strings

A string is a sequence of characters surrounded by double quotes; strings containing double quotes must represent the double quote using two consecutive double quotes. For example "IML is ""fun"" to learn!" represents the string —> IML is "fun" to learn.

3.3 Types

IML has five declarable types: integer, real, string, pixel, and image.

integer - 32 bit signed integer value
real - 64 bit signed floating point value
string - Printable sequence of characters
pixel - 32 bit value representing red, blue, and green color components; each channel uses only 8 bits, so 8 bits of this type are wasted.
image - Essentially and array of pixels; however, this type is more of a class, with some methods associated to it.

4.4 Conversions

The following conversion are valid between types.

3.4.1 Integer and Real

Integers may be converted to reals without loss of accuracy; reals may be converted to integers, any fractional part will be lost. Behavior is undefined if the value of the real exceeds 2^31-1.

3.4.2 Integer and Pixel

Integers and pixels may be freely converted, as may be useful for manual masking and manipulation of color channels.

3.5 Expressions

Expressions consist of identifiers (including function calls) and operators. The following list of operators are defined in IML, from highest precedence to lowest:

()	Grouping
-	Unary minus
ID	Indentifier/Function call
!	Logical NOT
~	Bitwise NOT
*	Multiplication
/	Division
%	Modulus
+	Addition
-	Subtraction
<<	Bitwise shift left
>>	Bitwise shift right
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Logical equality
!=	Logical inequality
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise OR

&& Logical AND
|| Logical OR

= Assignment

3.6 Declarations

Declarations of variables and functions are described in this section in a pseudo-ANTLR type notation; both are derived from Pascal syntax.

3.6.1 Variable Declarations

Variable declarations take the form of

var ID [**,** ID]* **:** type**;**

where ID is any valid identifier as described in section 2.2, and 'type' is any of the five declarable types described in section 3.

3.6.2 Function Declarations

Function declarations take the form of

function ID **(** (PARAM_LIST)? **) :** type

where PARAM_LIST takes the form of

PARAM (**,** PARAM)*

where PARAM is further defined as

ID **:** type

where ID is any valid identifier as described in section 2.2, and 'type' is any valid IML type as defined in section 3.

3.7 Statements

This section described the various statements permissible in IML.

3.7.1 Expressions

Expression statements, such as assignments or function calls, take the form of

expression **;**

3.7.2 Compound Statements

Sequences of statements can be grouped together to form a single logical statement by enclosing them in braces:

{ (statement **;**)* **}**

3.7.3 Conditional Statement

There are two forms of the conditional statement:

if (expression **) then** statement1

and

if (expression **) then** statement1 **else** statement2

In either case, 'expression' is evaluated, and if it is non-zero then 'statement1' is executed; in the latter case, if 'expression' evaluates to zero, then 'statement2' is executed. "Else ambiguity" is resolved by connecting the else with the nearest "elseless if".

3.7.4 While Statement

The while statement takes the form of

while (expression **)** statement

'expression' is evaluated at the beginning of each iteration, and statement is executed if the result is non-zero; iteration terminates when 'expression' evaluates to zero.

3.7.5 Do Statement

The do statement takes the form

do statement **while (** expression **) ;**

'expression' is evaluated at the end of each iteration, and statement is executed if the result is non-zero; iteration terminates when 'expression' evaluates to zero. 'statement' is guaranteed to execute at least one time.

3.7.6 For Statement

The for statement takes the form of

for (expr1 **;** expr2 **;** expr3 **)** statement

and is equivalent to

```
expr1;
while(expr2){
        statement;
        expr3;
}
```

3.7.7 Return statement

Functions return to their caller via the return statement, which takes the form

return (expression)? **;**

The value of expression will be returned to the caller, assuming the function is declared to return a value of matching type. It is an error for a function not declared as null to omit 'expression',

as well as it is an error for a function declared as null to include 'expression'.

3.7.8 Print Statement

The print statement takes the form of

print expression (**,** expression)* **;**

Each expression will be evaluated and output on the same line.

3.8 Scope rules

IML allows for global variable declarations between the program statement and the first function declaration; these globals will be available to every function so long as such function does not declare a local variable by the same name. Variables declared within a function, including parameters, are local to that function and can not be accessed via any other function; such variables can have the same name as global variables, thereby making the global inaccessible within that function.

3.9 Program structure

The overall structure of an IML program is as follows:

program ID **;**
<global variable block>
<function definition block>
main()
end.

Program execution always begins at the first statement in the **main** function.

3.9.1 Example

The following sample program demonstrates how to invert an image

```
program InvertImage:

function InvertPixel(p : pixel) : pixel
{
        var np : pixel;
        np.red = 255 - p.red;
        np.green = 255 - p.green;
        np.blue = 255 - p.blue;
        return np;
}

main()
{
        var img : image;
        var j, k : integer;
        img = image::open("myimage.jpg");

        for(j=0; j<img.height; j = j + 1){
                for(k=0; k<img.width; k = k + 1){
```

```
                        img[k,j] = InvertPixel(img[k,j]);
                    }
                }

                image::save(img, "myimage_neg.jpg");
        }

        end.
```

4. Project Plan

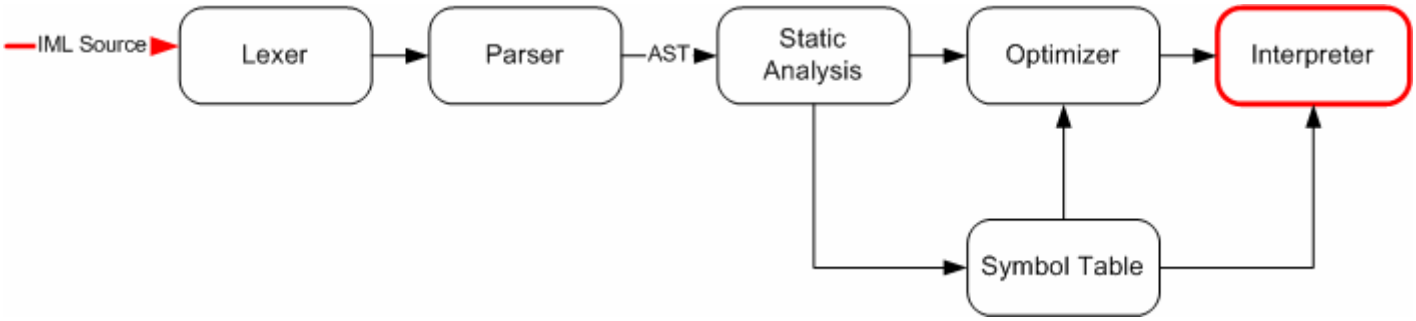
4.1 Planning and Specification

It was the author's intention to design a language which had a familiar look and feel but was distinct enough to not be just another C or Java knock-off; the use of Pascal style function and variable declarations (as well as the leading "program" and tailing "end." statement) is a tribute to the author's favorite language. The overall language took form with the creation of several small sample programs designed to illustrate the language. Formal specification of the language was finalized during the writing of the language reference manual.

4.2 Environment and Tools

All development was done under FreeBSD 6.0, running the K Desktop Environment (KDE); all source files were editing using Kate (K Advanced Text Editor). Antlr version 2.7.6 was used to generate the lexer and parser from the grammar specification, and a very unstable release of Sun's Java JDK 1.5 was used to compile and test all java source.

5. Architectural Design



5.1 Lexer

The Antlr generated lexer converts the input stream into a series of tokens to be processed by the parser.

5.2 Parser

The Antlr generated parser checks that the syntax of the input matches that of the grammar, and constructs an abstract syntax tree from the lexed input stream.

5.3 Static Analysis

The static analysis stage is the first walk of the AST, during which, name resolution and type checking occur. and symbol tables are generated.

5.4 Optimizer

The optimizer is the second walk of the AST, and performs only simple constant folding.

5.5 Interpreter

The interpreter stage, predictably, executes the AST.

6. Lessons Learned

This project was surprisingly fun to work on; while I did not particularly care for the Antlr tool (most likely because I was too impatient to take the time to really learn it), when the time finally came that my own IML programs were running (and doing what I wanted them to do), there was a great sense of satisfaction.

My advice to future students is this:

1. Take the time to learn Antlr; it is a very powerful tool that greatly simplifies the project if used properly. (hint: don't look to my project for any stunning insight!)

2. Have fun with it! It is much easier than it sounds on the first day of class, and satisfaction when you're programs run is sure to make you smile.

7. Appendix

FILE : IML.g

```
class IMLLexer extends Lexer;
options
{
        k = 2;
        charVocabulary = '\3'..'\377';
        exportVocab = IML;
        testLiterals = false;
}

PLUS        : '+';
MINUS               : '-';
MULT        : '*';
DIV         : '/';
MOD         : '%';
EQUALS              : "==";
NOT_EQ              : "!=";
AND         : "&&";
OR          : "||";
NOT         : '!';
GT          : '>';
LT          : '<';
GTE         : ">=";
LTE         : "<=";
SHIFTL              : "<<";
SHIFTR              : ">>";
```

```
BW_AND              : '&';
BW_OR               : '|';
BW_NOT              : '~';
BW_XOR              : '^';
ASSIGN              : '=';
SEMI          : ';';
COLON               : ':';
COMMA               : ',';
DOT           : '.';
STATIC              : "::";
PROPERTY  : "=>";

LPAREN      options { testLiterals = true; }        : '(';
RPAREN      options { testLiterals = true; }        : ')';
LBRACE              : '{';
RBRACE              : '}';
LBRACKET  : '[';
RBRACKET  : ']';

protected UNDERSCORE  : '_';
protected CHAR              : ('a'..'z' | 'A'..'Z');
protected DIGIT             : '0'..'9';
protected SPACE             : ' ';
protected LF          : '\n';
protected CR          : '\r';
protected TAB               : '\t';

protected REAL_T            : ;
protected INTEGER_T       : ;
protected STRING_T        : ;

ID options { testLiterals = true; }    : CHAR (CHAR | DIGIT | UNDERSCORE)* ;
NUMBER                                  : (DIGIT)+ ((DOT (DIGIT)* { $setType(REAL_T); }) | {
$setType(INTEGER_T); }) ;
STRING                                  : ""! (""! "" | ~(""))* ""! { $setType(STRING_T); };
WHITESPACE                              : ( SPACE | TAB | CR | LF { newline(); } ) {
$setType(Token.SKIP); } ;
COMMENT                                 : "/*"
                                        (
                                                options { greedy = false; }: (
                                                        (CR LF) => CR LF { newline(); } |
                                                        CR { newline(); }               |
                                                        LF { newline(); }               |
                                                        ~( '\n' | '\r' )
                                                )
                                        )*
                                    "*/" { $setType(Token.SKIP); } ;

class IMLParser extends Parser;
options
{
        k = 2;
        buildAST = true;
        exportVocab=IML;
```

```
}

tokens {
        VARS;
        PARAMS;
        PARAM;
        FBODY;
        IFBODY;
}

file            : "program"^ ID COLON!
                  (var_decl)*
                  (func_decl)*
                  main_decl
                  "end"! DOT!
                  EOF!
                ;

/* Private types */
private var_decl_list : ID (COMMA! ID)* { #var_decl_list = #([VARS, "VARS"], var_decl_list); } ;
private type          : "image" | "pixel" | "integer" | "real" | "string" | "null" ;
private var_decl      : "var"^ var_decl_list COLON! type SEMI! ;
private param_decl0         : ID COLON! type { #param_decl0 = #([PARAM], param_decl0); } ;
private param_decl  : (param_decl0 (COMMA! param_decl0)*) { #param_decl = #([PARAMS,
"PARAMS"], param_decl); }
                    | /* Nothing */ { #param_decl = #([PARAMS, "PARAMS"], param_decl); } ;
private func_body   : (var_decl)* (statement)* { #func_body = #([FBODY, "FUNCTION BODY"],
func_body); } ;
private function_call : ID^ LPAREN! (expr (COMMA! expr)*)? RPAREN! ;
private lvalue        : ID ( (DOT^ (pixel_property | image_property)) | (LBRACKET^ expr (COMMA!
expr)* RBRACKET!) )? ;
private static_method       : type STATIC^ image_method (LPAREN! (expr (COMMA! expr)*)?
RPAREN!)? ;
private pixel_property      : "red" | "green" | "blue" ;
private image_property      : "width" | "height" ;
private image_method        : "create" | "load" | "save" ;

func_decl     : "function"^ ID LPAREN! (param_decl) RPAREN! COLON! type LBRACE! func_body
RBRACE! ;
main_decl     : "main"^ LPAREN! RPAREN! LBRACE! func_body RBRACE! ;

statement     : if_statement                    |
                for_statement                   |
                while_statement           |
                do_statement                    |
                return_statement          |
                print_statement           |
                LBRACE^ (statement)* RBRACE!      |
                expr SEMI!
              ;

if_statement  : "if"^ LPAREN! expr RPAREN! statement (options { greedy=true; }: "else"! statement)? ;
for_statement       : "for"^ LPAREN! expr SEMI! expr SEMI! expr RPAREN! statement ;
while_statement : "while"^ LPAREN! expr RPAREN! statement ;
```

```
do_statement        : "do"^ statement "while"! LPAREN! expr RPAREN! SEMI! ;
return_statement: "return"^ expr SEMI! ;
print_statement     : "print"^ (expr (COMMA! expr)*)? SEMI! ;

expr            : expr0 (ASSIGN^ expr0)* ;
expr0           : expr1 (OR^ expr1)* ;
expr1           : expr2 (AND^ expr2)* ;
expr2           : expr3 (BW_OR^ expr3)* ;
expr3           : expr4 (BW_XOR^ expr4)* ;
expr4           : expr5 (BW_AND^ expr5)* ;
expr5           : expr6 ((EQUALS^ | NOT_EQ^) expr6)* ;
expr6           : expr7 ((GT^ | GTE^ | LT^ | LTE^) expr7)* ;
expr7           : expr8 ((SHIFTL^ | SHIFTR^) expr8)* ;
expr8           : expr9 ((PLUS^ | MINUS^) expr9)* ;
expr9           : expr10 ((MULT^ | DIV^ | MOD^) expr10)* ;
expr10              : ((BW_NOT^)? expr11) ;
expr11              : ((NOT^)? expr12) ;
expr12              : function_call                 |
                  static_method                 |
                  INTEGER_T                     |
                  REAL_T                |
                  STRING_T              |
                  MINUS^ expr11         |
                  lvalue                |
                  LPAREN! expr0 RPAREN!
                ;
```

FILE: IMLLexer.java

```
// $ANTLR 2.7.6 (20060127): "IML.g" -> "IMLLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;
```

```
public class IMLLexer extends antlr.CharScanner implements IMLTokenTypes, TokenStream
 {
public IMLLexer(InputStream in) {
	this(new ByteBuffer(in));
}
public IMLLexer(Reader in) {
	this(new CharBuffer(in));
}
public IMLLexer(InputBuffer ib) {
	this(new LexerSharedInputState(ib));
}
public IMLLexer(LexerSharedInputState state) {
	super(state);
	caseSensitiveLiterals = true;
	setCaseSensitive(true);
	literals = new Hashtable();
	literals.put(new ANTLRHashString("main", this), new Integer(75));
	literals.put(new ANTLRHashString("for", this), new Integer(78));
	literals.put(new ANTLRHashString("print", this), new Integer(82));
	literals.put(new ANTLRHashString("create", this), new Integer(71));
	literals.put(new ANTLRHashString("integer", this), new Integer(61));
	literals.put(new ANTLRHashString("end", this), new Integer(58));
	literals.put(new ANTLRHashString("string", this), new Integer(63));
	literals.put(new ANTLRHashString("load", this), new Integer(72));
	literals.put(new ANTLRHashString("blue", this), new Integer(68));
	literals.put(new ANTLRHashString("image", this), new Integer(59));
	literals.put(new ANTLRHashString("do", this), new Integer(80));
	literals.put(new ANTLRHashString("null", this), new Integer(64));
	literals.put(new ANTLRHashString("function", this), new Integer(74));
	literals.put(new ANTLRHashString("width", this), new Integer(69));
	literals.put(new ANTLRHashString("while", this), new Integer(79));
	literals.put(new ANTLRHashString("real", this), new Integer(62));
	literals.put(new ANTLRHashString("height", this), new Integer(70));
	literals.put(new ANTLRHashString("return", this), new Integer(81));
	literals.put(new ANTLRHashString("if", this), new Integer(76));
	literals.put(new ANTLRHashString("program", this), new Integer(57));
	literals.put(new ANTLRHashString("pixel", this), new Integer(60));
	literals.put(new ANTLRHashString("save", this), new Integer(73));
	literals.put(new ANTLRHashString("else", this), new Integer(77));
	literals.put(new ANTLRHashString("green", this), new Integer(67));
	literals.put(new ANTLRHashString("var", this), new Integer(65));
	literals.put(new ANTLRHashString("red", this), new Integer(66));
}

public Token nextToken() throws TokenStreamException {
	Token theRetToken=null;
tryAgain:
	for (;;) {
		Token _token = null;
		int _ttype = Token.INVALID_TYPE;
		resetText();
		try {   // for char stream error handling
			try {   // for lexical error handling
```

```
switch ( LA(1)) {
case '+':
{
        mPLUS(true);
        theRetToken=_returnToken;
        break;
}
case '-':
{
        mMINUS(true);
        theRetToken=_returnToken;
        break;
}
case '*':
{
        mMULT(true);
        theRetToken=_returnToken;
        break;
}
case '%':
{
        mMOD(true);
        theRetToken=_returnToken;
        break;
}
case '~':
{
        mBW_NOT(true);
        theRetToken=_returnToken;
        break;
}
case '^':
{
        mBW_XOR(true);
        theRetToken=_returnToken;
        break;
}
case ';':
{
        mSEMI(true);
        theRetToken=_returnToken;
        break;
}
case ',':
{
        mCOMMA(true);
        theRetToken=_returnToken;
        break;
}
case '.':
{
        mDOT(true);
        theRetToken=_returnToken;
        break;
```

```
}
case '(':
{
        mLPAREN(true);
        theRetToken=_returnToken;
        break;
}
case ')':
{
        mRPAREN(true);
        theRetToken=_returnToken;
        break;
}
case '{':
{
        mLBRACE(true);
        theRetToken=_returnToken;
        break;
}
case '}':
{
        mRBRACE(true);
        theRetToken=_returnToken;
        break;
}
case '[':
{
        mLBRACKET(true);
        theRetToken=_returnToken;
        break;
}
case ']':
{
        mRBRACKET(true);
        theRetToken=_returnToken;
        break;
}
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case 'a':  case 'b':
case 'c':  case 'd':  case 'e':  case 'f':
case 'g':  case 'h':  case 'i':  case 'j':
case 'k':  case 'l':  case 'm':  case 'n':
case 'o':  case 'p':  case 'q':  case 'r':
case 's':  case 't':  case 'u':  case 'v':
case 'w':  case 'x':  case 'y':  case 'z':
{
        mID(true);
        theRetToken=_returnToken;
        break;
```

```
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
        mNUMBER(true);
        theRetToken=_returnToken;
        break;
}
case '"':
{
        mSTRING(true);
        theRetToken=_returnToken;
        break;
}
case '\t':  case '\n':  case '\r':  case ' ':
{
        mWHITESPACE(true);
        theRetToken=_returnToken;
        break;
}
default:
        if ((LA(1)=='=') && (LA(2)=='=')) {
                mEQUALS(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='!') && (LA(2)=='=')) {
                mNOT_EQ(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='&') && (LA(2)=='&')) {
                mAND(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='|') && (LA(2)=='|')) {
                mOR(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='>') && (LA(2)=='=')) {
                mGTE(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='<') && (LA(2)=='=')) {
                mLTE(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='<') && (LA(2)=='<')) {
                mSHIFTL(true);
                theRetToken=_returnToken;
        }
        else if ((LA(1)=='>') && (LA(2)=='>')) {
                mSHIFTR(true);
                theRetToken=_returnToken;
        }
```

```
                                        else if ((LA(1)==':') && (LA(2)==':')) {
                                                mSTATIC(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='=') && (LA(2)=='>')) {
                                                mPROPERTY(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='/') && (LA(2)=='*')) {
                                                mCOMMENT(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='/') && (true)) {
                                                mDIV(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='!') && (true)) {
                                                mNOT(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='>') && (true)) {
                                                mGT(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='<') && (true)) {
                                                mLT(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='&') && (true)) {
                                                mBW_AND(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='|') && (true)) {
                                                mBW_OR(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)=='=') && (true)) {
                                                mASSIGN(true);
                                                theRetToken=_returnToken;
                                        }
                                        else if ((LA(1)==':') && (true)) {
                                                mCOLON(true);
                                                theRetToken=_returnToken;
                                        }
                                else {
                                        if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken =
makeToken(Token.EOF_TYPE);}
                                else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
                                }
                                }
                                if ( _returnToken==null ) continue tryAgain; // found SKIP token
                                _ttype = _returnToken.getType();
                                _returnToken.setType(_ttype);
```

```
                        return _returnToken;
                }
                catch (RecognitionException e) {
                        throw new TokenStreamRecognitionException(e);
                }
        }
        catch (CharStreamException cse) {
                if ( cse instanceof CharStreamIOException ) {
                        throw new TokenStreamIOException(((CharStreamIOException)cse).io);
                }
                else {
                        throw new TokenStreamException(cse.getMessage());
                }
        }
    }
}

    public final void mPLUS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = PLUS;
        int _saveIndex;

        match('+');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mMINUS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = MINUS;
        int _saveIndex;

        match('-');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mMULT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = MULT;
        int _saveIndex;

        match('*');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
```

```
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mDIV(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIV;
            int _saveIndex;

            match('/');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mMOD(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MOD;
            int _saveIndex;

            match('%');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mEQUALS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = EQUALS;
            int _saveIndex;

            match("==");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mNOT_EQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NOT_EQ;
            int _saveIndex;

            match("!=");
```

```
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mAND(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = AND;
		int _saveIndex;

		match("&&");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mOR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = OR;
		int _saveIndex;

		match("||");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mNOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NOT;
		int _saveIndex;

		match('!');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mGT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GT;
		int _saveIndex;
```

```
		match('>');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LT;
		int _saveIndex;

		match('<');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mGTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GTE;
		int _saveIndex;

		match(">=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LTE;
		int _saveIndex;

		match("<=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSHIFTL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
```

```
		_ttype = SHIFTL;
		int _saveIndex;

		match("<<");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSHIFTR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SHIFTR;
		int _saveIndex;

		match(">>");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mBW_AND(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = BW_AND;
		int _saveIndex;

		match('&');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mBW_OR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = BW_OR;
		int _saveIndex;

		match('|');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mBW_NOT(boolean _createToken) throws RecognitionException,
```

```
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = BW_NOT;
		int _saveIndex;

		match('~');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mBW_XOR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = BW_XOR;
		int _saveIndex;

		match('^');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ASSIGN;
		int _saveIndex;

		match('=');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSEMI(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SEMI;
		int _saveIndex;

		match(';');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mCOLON(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COLON;
		int _saveIndex;

		match(':');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mCOMMA(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COMMA;
		int _saveIndex;

		match(',');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mDOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DOT;
		int _saveIndex;

		match('.');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSTATIC(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STATIC;
		int _saveIndex;

		match("::");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
```

```
		_returnToken = _token;
	}

	public final void mPROPERTY(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = PROPERTY;
		int _saveIndex;

		match("=>");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLPAREN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LPAREN;
		int _saveIndex;

		match('(');
		_ttype = testLiteralsTable(_ttype);
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRPAREN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RPAREN;
		int _saveIndex;

		match(')');
		_ttype = testLiteralsTable(_ttype);
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLBRACE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LBRACE;
		int _saveIndex;

		match('{');
```

```
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRBRACE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RBRACE;
		int _saveIndex;

		match('}');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLBRACKET(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LBRACKET;
		int _saveIndex;

		match('[');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRBRACKET(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RBRACKET;
		int _saveIndex;

		match(']');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mUNDERSCORE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = UNDERSCORE;
		int _saveIndex;
```

```
		match('_');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mCHAR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = CHAR;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'a':  case 'b':  case 'c':  case 'd':
		case 'e':  case 'f':  case 'g':  case 'h':
		case 'i':  case 'j':  case 'k':  case 'l':
		case 'm':  case 'n':  case 'o':  case 'p':
		case 'q':  case 'r':  case 's':  case 't':
		case 'u':  case 'v':  case 'w':  case 'x':
		case 'y':  case 'z':
		{
			matchRange('a','z');
			break;
		}
		case 'A':  case 'B':  case 'C':  case 'D':
		case 'E':  case 'F':  case 'G':  case 'H':
		case 'I':  case 'J':  case 'K':  case 'L':
		case 'M':  case 'N':  case 'O':  case 'P':
		case 'Q':  case 'R':  case 'S':  case 'T':
		case 'U':  case 'V':  case 'W':  case 'X':
		case 'Y':  case 'Z':
		{
			matchRange('A','Z');
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
		}
		}
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mDIGIT(boolean _createToken) throws RecognitionException,
```

```
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DIGIT;
		int _saveIndex;

		matchRange('0','9');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mSPACE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SPACE;
		int _saveIndex;

		match(' ');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mLF(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LF;
		int _saveIndex;

		match('\n');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mCR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = CR;
		int _saveIndex;

		match('\r');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	protected final void mTAB(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = TAB;
		int _saveIndex;

		match('\t');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mREAL_T(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = REAL_T;
		int _saveIndex;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mINTEGER_T(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = INTEGER_T;
		int _saveIndex;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mSTRING_T(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STRING_T;
		int _saveIndex;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mID(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ID;
		int _saveIndex;

		mCHAR(false);
		{
		_loop47:
		do {
			switch ( LA(1)) {
			case 'A':  case 'B':  case 'C':  case 'D':
			case 'E':  case 'F':  case 'G':  case 'H':
			case 'I':  case 'J':  case 'K':  case 'L':
			case 'M':  case 'N':  case 'O':  case 'P':
			case 'Q':  case 'R':  case 'S':  case 'T':
			case 'U':  case 'V':  case 'W':  case 'X':
			case 'Y':  case 'Z':  case 'a':  case 'b':
			case 'c':  case 'd':  case 'e':  case 'f':
			case 'g':  case 'h':  case 'i':  case 'j':
			case 'k':  case 'l':  case 'm':  case 'n':
			case 'o':  case 'p':  case 'q':  case 'r':
			case 's':  case 't':  case 'u':  case 'v':
			case 'w':  case 'x':  case 'y':  case 'z':
			{
				mCHAR(false);
				break;
			}
			case '0':  case '1':  case '2':  case '3':
			case '4':  case '5':  case '6':  case '7':
			case '8':  case '9':
			{
				mDIGIT(false);
				break;
			}
			case '_':
			{
				mUNDERSCORE(false);
				break;
			}
			default:
			{
				break _loop47;
			}
			}
		} while (true);
		}
		_ttype = testLiteralsTable(_ttype);
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mNUMBER(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NUMBER;
		int _saveIndex;

		{
		int _cnt50=0;
		_loop50:
		do {
			if (((LA(1) >= '0' && LA(1) <= '9'))) {
				mDIGIT(false);
			}
			else {
				if ( _cnt50>=1 ) { break _loop50; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
			}

			_cnt50++;
		} while (true);
		}
		{
		if ((LA(1)=='.')) {
			{
			mDOT(false);
			{
			_loop54:
			do {
				if (((LA(1) >= '0' && LA(1) <= '9'))) {
					mDIGIT(false);
				}
				else {
					break _loop54;
				}

			} while (true);
			}
			if ( inputState.guessing==0 ) {
				_ttype = REAL_T;
			}
			}
		}
		else {
			if ( inputState.guessing==0 ) {
				_ttype = INTEGER_T;
			}
		}

		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
```

```
		_returnToken = _token;
	}

	public final void mSTRING(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STRING;
		int _saveIndex;

		_saveIndex=text.length();
		match("");
		text.setLength(_saveIndex);
		{
		_loop58:
		do {
			if ((LA(1)=="") && (LA(2)="")) {
				_saveIndex=text.length();
				match("");
				text.setLength(_saveIndex);
				match("");
			}
			else if ((_tokenSet_0.member(LA(1)))) {
				{
				match(_tokenSet_0);
				}
			}
			else {
				break _loop58;
			}

		} while (true);
		}
		_saveIndex=text.length();
		match("");
		text.setLength(_saveIndex);
		if ( inputState.guessing==0 ) {
			_ttype = STRING_T;
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mWHITESPACE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = WHITESPACE;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case ' ':
```

```
		{
			mSPACE(false);
			break;
		}
		case '\t':
		{
			mTAB(false);
			break;
		}
		case '\r':
		{
			mCR(false);
			break;
		}
		case '\n':
		{
			mLF(false);
			if ( inputState.guessing==0 ) {
				newline();
			}
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
		}
		}
		}
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP;
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mCOMMENT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COMMENT;
		int _saveIndex;

		match("/*");
		{
		_loop67:
		do {
			// nongreedy exit test
			if ((LA(1)=='*') && (LA(2)=='/')) break _loop67;
			if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && ((LA(2) >= '\u0003' && LA(2) <=
'\u00ff'))) {
				{
```

```
                                boolean synPredMatched65 = false;
                                if (((LA(1)=='\r') && (LA(2)=='\n'))) {
                                        int _m65 = mark();
                                        synPredMatched65 = true;
                                        inputState.guessing++;
                                        try {
                                                {
                                                mCR(false);
                                                mLF(false);
                                                }
                                        }
                                        catch (RecognitionException pe) {
                                                synPredMatched65 = false;
                                        }
                                        rewind(_m65);
inputState.guessing--;
                                }
                                if ( synPredMatched65 ) {
                                        mCR(false);
                                        mLF(false);
                                        if ( inputState.guessing==0 ) {
                                                newline();
                                        }
                                }
                                else if ((LA(1)=='\r') && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                                        mCR(false);
                                        if ( inputState.guessing==0 ) {
                                                newline();
                                        }
                                }
                                else if ((LA(1)=='\n')) {
                                        mLF(false);
                                        if ( inputState.guessing==0 ) {
                                                newline();
                                        }
                                }
                                else if ((_tokenSet_1.member(LA(1)))) {
                                        {
                                        match(_tokenSet_1);
                                        }
                                }
                                else {
                                        throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                                }

                                }
                        }
                        else {
                                break _loop67;
                        }

                } while (true);
                }
```

```
		match("*/");
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP;
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}


	private static final long[] mk_tokenSet_0() {
		long[] data = new long[8];
		data[0]=-17179869192L;
		for (int i = 1; i<=3; i++) { data[i]=-1L; }
		return data;
	}
	public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
	private static final long[] mk_tokenSet_1() {
		long[] data = new long[8];
		data[0]=-9224L;
		for (int i = 1; i<=3; i++) { data[i]=-1L; }
		return data;
	}
	public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());

	}
```

FILE: IMLParser.java

```
// $ANTLR 2.7.6 (20060127): "IML.g" -> "IMLParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class IMLParser extends antlr.LLkParser       implements IMLTokenTypes
 {
```

```
protected IMLParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public IMLParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected IMLParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public IMLParser(TokenStream lexer) {
  this(lexer,2);
}

public IMLParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

        public final void file() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST file_AST = null;

                try {      // for error handling
                        AST tmp1_AST = null;
                        tmp1_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp1_AST);
                        match(LITERAL_program);
                        AST tmp2_AST = null;
                        tmp2_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp2_AST);
                        match(ID);
                        match(COLON);
                        {
                        _loop70:
                        do {
                                if ((LA(1)==LITERAL_var)) {
                                        var_decl();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
```

```
                        break _loop70;
                    }

                } while (true);
                }
                {
                _loop72:
                do {
                    if ((LA(1)==LITERAL_function)) {
                        func_decl();
                        astFactory.addASTChild(currentAST, returnAST);
                    }
                    else {
                        break _loop72;
                    }

                } while (true);
                }
                main_decl();
                astFactory.addASTChild(currentAST, returnAST);
                match(LITERAL_end);
                match(DOT);
                match(Token.EOF_TYPE);
                file_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_0);
            }
            returnAST = file_AST;
    }

    private final void var_decl() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST var_decl_AST = null;

            try {      // for error handling
                AST tmp7_AST = null;
                tmp7_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp7_AST);
                match(LITERAL_var);
                var_decl_list();
                astFactory.addASTChild(currentAST, returnAST);
                match(COLON);
                type();
                astFactory.addASTChild(currentAST, returnAST);
                match(SEMI);
                var_decl_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_1);
```

```
        }
        returnAST = var_decl_AST;
}

public final void func_decl() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST func_decl_AST = null;

        try {      // for error handling
                AST tmp10_AST = null;
                tmp10_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp10_AST);
                match(LITERAL_function);
                AST tmp11_AST = null;
                tmp11_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp11_AST);
                match(ID);
                match(LPAREN);
                {
                param_decl();
                astFactory.addASTChild(currentAST, returnAST);
                }
                match(RPAREN);
                match(COLON);
                type();
                astFactory.addASTChild(currentAST, returnAST);
                match(LBRACE);
                func_body();
                astFactory.addASTChild(currentAST, returnAST);
                match(RBRACE);
                func_decl_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_2);
        }
        returnAST = func_decl_AST;
}

public final void main_decl() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST main_decl_AST = null;

        try {      // for error handling
                AST tmp17_AST = null;
                tmp17_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp17_AST);
                match(LITERAL_main);
                match(LPAREN);
                match(RPAREN);
```

```
                match(LBRACE);
                func_body();
                astFactory.addASTChild(currentAST, returnAST);
                match(RBRACE);
                main_decl_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_3);
        }
        returnAST = main_decl_AST;
}

private final void var_decl_list() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST var_decl_list_AST = null;

        try {      // for error handling
                AST tmp22_AST = null;
                tmp22_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp22_AST);
                match(ID);
                {
                _loop75:
                do {
                        if ((LA(1)==COMMA)) {
                                match(COMMA);
                                AST tmp24_AST = null;
                                tmp24_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp24_AST);
                                match(ID);
                        }
                        else {
                                break _loop75;
                        }

                } while (true);
                }
                var_decl_list_AST = (AST)currentAST.root;
                var_decl_list_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(VARS,"VARS")).add(var_decl_list_AST));
                currentAST.root = var_decl_list_AST;
                currentAST.child = var_decl_list_AST!=null
&&var_decl_list_AST.getFirstChild()!=null ?
                        var_decl_list_AST.getFirstChild() : var_decl_list_AST;
                currentAST.advanceChildToEnd();
                var_decl_list_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_4);
        }
```

```
		returnAST = var_decl_list_AST;
}

private final void type() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST type_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case LITERAL_image:
			{
				AST tmp25_AST = null;
				tmp25_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp25_AST);
				match(LITERAL_image);
				type_AST = (AST)currentAST.root;
				break;
			}
			case LITERAL_pixel:
			{
				AST tmp26_AST = null;
				tmp26_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp26_AST);
				match(LITERAL_pixel);
				type_AST = (AST)currentAST.root;
				break;
			}
			case LITERAL_integer:
			{
				AST tmp27_AST = null;
				tmp27_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp27_AST);
				match(LITERAL_integer);
				type_AST = (AST)currentAST.root;
				break;
			}
			case LITERAL_real:
			{
				AST tmp28_AST = null;
				tmp28_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp28_AST);
				match(LITERAL_real);
				type_AST = (AST)currentAST.root;
				break;
			}
			case LITERAL_string:
			{
				AST tmp29_AST = null;
				tmp29_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp29_AST);
				match(LITERAL_string);
				type_AST = (AST)currentAST.root;
```

```
                        break;
                    }
                    case LITERAL_null:
                    {
                        AST tmp30_AST = null;
                        tmp30_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp30_AST);
                        match(LITERAL_null);
                        type_AST = (AST)currentAST.root;
                        break;
                    }
                    default:
                    {
                        throw new NoViableAltException(LT(1), getFilename());
                    }
                    }
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    recover(ex,_tokenSet_5);
                }
                returnAST = type_AST;
        }

        private final void param_decl0() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST param_decl0_AST = null;

                try {      // for error handling
                    AST tmp31_AST = null;
                    tmp31_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp31_AST);
                    match(ID);
                    match(COLON);
                    type();
                    astFactory.addASTChild(currentAST, returnAST);
                    param_decl0_AST = (AST)currentAST.root;
                    param_decl0_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PARAM)).add(param_decl0_AST));
                    currentAST.root = param_decl0_AST;
                    currentAST.child = param_decl0_AST!=null
&&param_decl0_AST.getFirstChild()!=null ?
                        param_decl0_AST.getFirstChild() : param_decl0_AST;
                    currentAST.advanceChildToEnd();
                    param_decl0_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    recover(ex,_tokenSet_6);
                }
                returnAST = param_decl0_AST;
        }
```

```
	private final void param_decl() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST param_decl_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case ID:
			{
				{
				param_decl0();
				astFactory.addASTChild(currentAST, returnAST);
				{
				_loop82:
				do {
					if ((LA(1)==COMMA)) {
						match(COMMA);
						param_decl0();
						astFactory.addASTChild(currentAST, returnAST);
					}
					else {
						break _loop82;
					}

				} while (true);
				}
				}
				param_decl_AST = (AST)currentAST.root;
				param_decl_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PARAMS,"PARAMS")).add(param_decl_AST));
				currentAST.root = param_decl_AST;
				currentAST.child = param_decl_AST!=null
&&param_decl_AST.getFirstChild()!=null ?
					param_decl_AST.getFirstChild() : param_decl_AST;
				currentAST.advanceChildToEnd();
				param_decl_AST = (AST)currentAST.root;
				break;
			}
			case RPAREN:
			{
				param_decl_AST = (AST)currentAST.root;
				param_decl_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PARAMS,"PARAMS")).add(param_decl_AST));
				currentAST.root = param_decl_AST;
				currentAST.child = param_decl_AST!=null
&&param_decl_AST.getFirstChild()!=null ?
					param_decl_AST.getFirstChild() : param_decl_AST;
				currentAST.advanceChildToEnd();
				param_decl_AST = (AST)currentAST.root;
				break;
			}
			default:
```

```
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_7);
        }
        returnAST = param_decl_AST;
    }

    private final void func_body() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST func_body_AST = null;

        try {      // for error handling
                {
                _loop85:
                do {
                        if ((LA(1)==LITERAL_var)) {
                                var_decl();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop85;
                        }

                } while (true);
                }
                {
                _loop87:
                do {
                        if ((_tokenSet_8.member(LA(1)))) {
                                statement();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop87;
                        }

                } while (true);
                }
                func_body_AST = (AST)currentAST.root;
                func_body_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(FBODY,"FUNCTION BODY")).add(func_body_AST));
                currentAST.root = func_body_AST;
                currentAST.child = func_body_AST!=null &&func_body_AST.getFirstChild()!=null
?
                        func_body_AST.getFirstChild() : func_body_AST;
                currentAST.advanceChildToEnd();
                func_body_AST = (AST)currentAST.root;
```

```
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_9);
	}
	returnAST = func_body_AST;
}

public final void statement() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST statement_AST = null;

	try {	// for error handling
		switch ( LA(1)) {
		case LITERAL_if:
		{
			if_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_for:
		{
			for_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_while:
		{
			while_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_do:
		{
			do_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_return:
		{
			return_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_print:
		{
			print_statement();
```

```
                astFactory.addASTChild(currentAST, returnAST);
                statement_AST = (AST)currentAST.root;
                break;
        }
        case LBRACE:
        {
                AST tmp34_AST = null;
                tmp34_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp34_AST);
                match(LBRACE);
                {
                _loop112:
                do {
                        if ((_tokenSet_8.member(LA(1)))) {
                                statement();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop112;
                        }

                } while (true);
                }
                match(RBRACE);
                statement_AST = (AST)currentAST.root;
                break;
        }
        case MINUS:
        case NOT:
        case BW_NOT:
        case LPAREN:
        case REAL_T:
        case INTEGER_T:
        case STRING_T:
        case ID:
        case LITERAL_image:
        case LITERAL_pixel:
        case LITERAL_integer:
        case LITERAL_real:
        case LITERAL_string:
        case LITERAL_null:
        {
                expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(SEMI);
                statement_AST = (AST)currentAST.root;
                break;
        }
        default:
        {
                throw new NoViableAltException(LT(1), getFilename());
        }
        }
}
```

```
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_10);
	}
	returnAST = statement_AST;
}

private final void function_call() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST function_call_AST = null;

	try {	// for error handling
		AST tmp37_AST = null;
		tmp37_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp37_AST);
		match(ID);
		match(LPAREN);
		{
		switch ( LA(1)) {
		case MINUS:
		case NOT:
		case BW_NOT:
		case LPAREN:
		case REAL_T:
		case INTEGER_T:
		case STRING_T:
		case ID:
		case LITERAL_image:
		case LITERAL_pixel:
		case LITERAL_integer:
		case LITERAL_real:
		case LITERAL_string:
		case LITERAL_null:
		{
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			{
			_loop91:
			do {
				if ((LA(1)==COMMA)) {
					match(COMMA);
					expr();
					astFactory.addASTChild(currentAST, returnAST);
				}
				else {
					break _loop91;
				}

			} while (true);
			}
			break;
		}
```

```
		case RPAREN:
		{
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		match(RPAREN);
		function_call_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = function_call_AST;
}

public final void expr() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST expr_AST = null;

	try {	// for error handling
		expr0();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop125:
		do {
			if ((LA(1)==ASSIGN)) {
				AST tmp41_AST = null;
				tmp41_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp41_AST);
				match(ASSIGN);
				expr0();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop125;
			}

		} while (true);
		}
		expr_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_12);
	}
	returnAST = expr_AST;
}
```

```
private final void lvalue() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST lvalue_AST = null;

	try {	// for error handling
		AST tmp42_AST = null;
		tmp42_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp42_AST);
		match(ID);
		{
		switch ( LA(1)) {
		case DOT:
		{
			{
			AST tmp43_AST = null;
			tmp43_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp43_AST);
			match(DOT);
			{
			switch ( LA(1)) {
			case LITERAL_red:
			case LITERAL_green:
			case LITERAL_blue:
			{
				pixel_property();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			case LITERAL_width:
			case LITERAL_height:
			{
				image_property();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1), getFilename());
			}
			}
			}
			}
			break;
		}
		case LBRACKET:
		{
			{
			AST tmp44_AST = null;
			tmp44_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp44_AST);
			match(LBRACKET);
```

```
                expr();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop98:
                do {
                        if ((LA(1)==COMMA)) {
                                match(COMMA);
                                expr();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop98;
                        }

                } while (true);
                }
                match(RBRACKET);
                }
                break;
        }
        case PLUS:
        case MINUS:
        case MULT:
        case DIV:
        case MOD:
        case EQUALS:
        case NOT_EQ:
        case AND:
        case OR:
        case GT:
        case LT:
        case GTE:
        case LTE:
        case SHIFTL:
        case SHIFTR:
        case BW_AND:
        case BW_OR:
        case BW_XOR:
        case ASSIGN:
        case SEMI:
        case COMMA:
        case RPAREN:
        case RBRACKET:
        {
                break;
        }
        default:
        {
                throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        lvalue_AST = (AST)currentAST.root;
}
```

```
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = lvalue_AST;
}

private final void pixel_property() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST pixel_property_AST = null;

	try {	// for error handling
		switch ( LA(1)) {
		case LITERAL_red:
		{
			AST tmp47_AST = null;
			tmp47_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp47_AST);
			match(LITERAL_red);
			pixel_property_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_green:
		{
			AST tmp48_AST = null;
			tmp48_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp48_AST);
			match(LITERAL_green);
			pixel_property_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_blue:
		{
			AST tmp49_AST = null;
			tmp49_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp49_AST);
			match(LITERAL_blue);
			pixel_property_AST = (AST)currentAST.root;
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = pixel_property_AST;
}
```

```
private final void image_property() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST image_property_AST = null;

	try {	// for error handling
		switch ( LA(1)) {
		case LITERAL_width:
		{
			AST tmp50_AST = null;
			tmp50_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp50_AST);
			match(LITERAL_width);
			image_property_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_height:
		{
			AST tmp51_AST = null;
			tmp51_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp51_AST);
			match(LITERAL_height);
			image_property_AST = (AST)currentAST.root;
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = image_property_AST;
}

private final void static_method() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST static_method_AST = null;

	try {	// for error handling
		type();
		astFactory.addASTChild(currentAST, returnAST);
		AST tmp52_AST = null;
		tmp52_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp52_AST);
		match(STATIC);
		image_method();
```

```
astFactory.addASTChild(currentAST, returnAST);
{
switch ( LA(1)) {
case LPAREN:
{
	match(LPAREN);
	{
	switch ( LA(1)) {
	case MINUS:
	case NOT:
	case BW_NOT:
	case LPAREN:
	case REAL_T:
	case INTEGER_T:
	case STRING_T:
	case ID:
	case LITERAL_image:
	case LITERAL_pixel:
	case LITERAL_integer:
	case LITERAL_real:
	case LITERAL_string:
	case LITERAL_null:
	{
		expr();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop103:
		do {
			if ((LA(1)==COMMA)) {
				match(COMMA);
				expr();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop103;
			}

		} while (true);
		}
		break;
	}
	case RPAREN:
	{
		break;
	}
	default:
	{
		throw new NoViableAltException(LT(1), getFilename());
	}
	}
	}
	match(RPAREN);
	break;
}
```

```
		case PLUS:
		case MINUS:
		case MULT:
		case DIV:
		case MOD:
		case EQUALS:
		case NOT_EQ:
		case AND:
		case OR:
		case GT:
		case LT:
		case GTE:
		case LTE:
		case SHIFTL:
		case SHIFTR:
		case BW_AND:
		case BW_OR:
		case BW_XOR:
		case ASSIGN:
		case SEMI:
		case COMMA:
		case RPAREN:
		case RBRACKET:
		{
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		static_method_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = static_method_AST;
}

private final void image_method() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST image_method_AST = null;

	try {	// for error handling
		switch ( LA(1)) {
		case LITERAL_create:
		{
			AST tmp56_AST = null;
			tmp56_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp56_AST);
```

```
                match(LITERAL_create);
                image_method_AST = (AST)currentAST.root;
                break;
            }
            case LITERAL_load:
            {
                AST tmp57_AST = null;
                tmp57_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp57_AST);
                match(LITERAL_load);
                image_method_AST = (AST)currentAST.root;
                break;
            }
            case LITERAL_save:
            {
                AST tmp58_AST = null;
                tmp58_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp58_AST);
                match(LITERAL_save);
                image_method_AST = (AST)currentAST.root;
                break;
            }
            default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
            }
        }
        catch (RecognitionException ex) {
            reportError(ex);
            recover(ex,_tokenSet_13);
        }
        returnAST = image_method_AST;
}

public final void if_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST if_statement_AST = null;

        try {      // for error handling
            AST tmp59_AST = null;
            tmp59_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp59_AST);
            match(LITERAL_if);
            match(LPAREN);
            expr();
            astFactory.addASTChild(currentAST, returnAST);
            match(RPAREN);
            statement();
            astFactory.addASTChild(currentAST, returnAST);
            {
            if ((LA(1)==LITERAL_else) && (_tokenSet_8.member(LA(2)))) {
```

```
                match(LITERAL_else);
                statement();
                astFactory.addASTChild(currentAST, returnAST);
            }
            else if ((_tokenSet_10.member(LA(1))) && (_tokenSet_14.member(LA(2)))) {
            }
            else {
                throw new NoViableAltException(LT(1), getFilename());
            }

            }
            if_statement_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
            reportError(ex);
            recover(ex,_tokenSet_10);
        }
        returnAST = if_statement_AST;
}

public final void for_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST for_statement_AST = null;

        try {      // for error handling
            AST tmp63_AST = null;
            tmp63_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp63_AST);
            match(LITERAL_for);
            match(LPAREN);
            expr();
            astFactory.addASTChild(currentAST, returnAST);
            match(SEMI);
            expr();
            astFactory.addASTChild(currentAST, returnAST);
            match(SEMI);
            expr();
            astFactory.addASTChild(currentAST, returnAST);
            match(RPAREN);
            statement();
            astFactory.addASTChild(currentAST, returnAST);
            for_statement_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
            reportError(ex);
            recover(ex,_tokenSet_10);
        }
        returnAST = for_statement_AST;
}

public final void while_statement() throws RecognitionException, TokenStreamException {
```

```
		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST while_statement_AST = null;

		try {	// for error handling
			AST tmp68_AST = null;
			tmp68_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp68_AST);
			match(LITERAL_while);
			match(LPAREN);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			statement();
			astFactory.addASTChild(currentAST, returnAST);
			while_statement_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			recover(ex,_tokenSet_10);
		}
		returnAST = while_statement_AST;
	}

	public final void do_statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST do_statement_AST = null;

		try {	// for error handling
			AST tmp71_AST = null;
			tmp71_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp71_AST);
			match(LITERAL_do);
			statement();
			astFactory.addASTChild(currentAST, returnAST);
			match(LITERAL_while);
			match(LPAREN);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			match(SEMI);
			do_statement_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			recover(ex,_tokenSet_10);
		}
		returnAST = do_statement_AST;
	}

	public final void return_statement() throws RecognitionException, TokenStreamException {
```

```
		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST return_statement_AST = null;

		try {	// for error handling
			AST tmp76_AST = null;
			tmp76_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp76_AST);
			match(LITERAL_return);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(SEMI);
			return_statement_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			recover(ex,_tokenSet_10);
		}
		returnAST = return_statement_AST;
}

public final void print_statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST print_statement_AST = null;

		try {	// for error handling
			AST tmp78_AST = null;
			tmp78_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp78_AST);
			match(LITERAL_print);
			{
			switch ( LA(1)) {
			case MINUS:
			case NOT:
			case BW_NOT:
			case LPAREN:
			case REAL_T:
			case INTEGER_T:
			case STRING_T:
			case ID:
			case LITERAL_image:
			case LITERAL_pixel:
			case LITERAL_integer:
			case LITERAL_real:
			case LITERAL_string:
			case LITERAL_null:
			{
				expr();
				astFactory.addASTChild(currentAST, returnAST);
				{
				_loop122:
				do {
```

```
                        if ((LA(1)==COMMA)) {
                                match(COMMA);
                                expr();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop122;
                        }

                } while (true);
                }
                break;
        }
        case SEMI:
        {
                break;
        }
        default:
        {
                throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        match(SEMI);
        print_statement_AST = (AST)currentAST.root;
    }
    catch (RecognitionException ex) {
        reportError(ex);
        recover(ex,_tokenSet_10);
    }
    returnAST = print_statement_AST;
}

public final void expr0() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST expr0_AST = null;

    try {      // for error handling
        expr1();
        astFactory.addASTChild(currentAST, returnAST);
        {
        _loop128:
        do {
                if ((LA(1)==OR)) {
                        AST tmp81_AST = null;
                        tmp81_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp81_AST);
                        match(OR);
                        expr1();
                        astFactory.addASTChild(currentAST, returnAST);
                }
                else {
```

```
                        break _loop128;
                    }

                } while (true);
                }
                expr0_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_15);
            }
            returnAST = expr0_AST;
        }

        public final void expr1() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST expr1_AST = null;

            try {      // for error handling
                expr2();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop131:
                do {
                    if ((LA(1)==AND)) {
                        AST tmp82_AST = null;
                        tmp82_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp82_AST);
                        match(AND);
                        expr2();
                        astFactory.addASTChild(currentAST, returnAST);
                    }
                    else {
                        break _loop131;
                    }

                } while (true);
                }
                expr1_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_16);
            }
            returnAST = expr1_AST;
        }

        public final void expr2() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST expr2_AST = null;
```

```
	try {	// for error handling
		expr3();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop134:
		do {
			if ((LA(1)==BW_OR)) {
				AST tmp83_AST = null;
				tmp83_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp83_AST);
				match(BW_OR);
				expr3();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop134;
			}

		} while (true);
		}
		expr2_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_17);
	}
	returnAST = expr2_AST;
}

public final void expr3() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST expr3_AST = null;

	try {	// for error handling
		expr4();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop137:
		do {
			if ((LA(1)==BW_XOR)) {
				AST tmp84_AST = null;
				tmp84_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp84_AST);
				match(BW_XOR);
				expr4();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop137;
			}
```

```
                } while (true);
                }
                expr3_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_18);
        }
        returnAST = expr3_AST;
}

public final void expr4() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expr4_AST = null;

        try {      // for error handling
                expr5();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop140:
                do {
                        if ((LA(1)==BW_AND)) {
                                AST tmp85_AST = null;
                                tmp85_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp85_AST);
                                match(BW_AND);
                                expr5();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop140;
                        }

                } while (true);
                }
                expr4_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_19);
        }
        returnAST = expr4_AST;
}

public final void expr5() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expr5_AST = null;

        try {      // for error handling
                expr6();
```

```
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop144:
                do {
                        if ((LA(1)==EQUALS||LA(1)==NOT_EQ)) {
                                {
                                switch ( LA(1)) {
                                case EQUALS:
                                {
                                        AST tmp86_AST = null;
                                        tmp86_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp86_AST);
                                        match(EQUALS);
                                        break;
                                }
                                case NOT_EQ:
                                {
                                        AST tmp87_AST = null;
                                        tmp87_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp87_AST);
                                        match(NOT_EQ);
                                        break;
                                }
                                default:
                                {
                                        throw new NoViableAltException(LT(1), getFilename());
                                }
                                }
                                }
                                expr6();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop144;
                        }

                } while (true);
                }
                expr5_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_20);
        }
        returnAST = expr5_AST;
}

public final void expr6() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expr6_AST = null;

        try {      // for error handling
```

```
expr7();
astFactory.addASTChild(currentAST, returnAST);
{
_loop148:
do {
	if (((LA(1) >= GT && LA(1) <= LTE))) {
		{
		switch ( LA(1)) {
		case GT:
		{
			AST tmp88_AST = null;
			tmp88_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp88_AST);
			match(GT);
			break;
		}
		case GTE:
		{
			AST tmp89_AST = null;
			tmp89_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp89_AST);
			match(GTE);
			break;
		}
		case LT:
		{
			AST tmp90_AST = null;
			tmp90_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp90_AST);
			match(LT);
			break;
		}
		case LTE:
		{
			AST tmp91_AST = null;
			tmp91_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp91_AST);
			match(LTE);
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		expr7();
		astFactory.addASTChild(currentAST, returnAST);
	}
	else {
		break _loop148;
	}

} while (true);
```

```
		}
		expr6_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_21);
	}
	returnAST = expr6_AST;
}

public final void expr7() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST expr7_AST = null;

	try {	// for error handling
		expr8();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop152:
		do {
			if ((LA(1)==SHIFTL||LA(1)==SHIFTR)) {
				{
				switch ( LA(1)) {
				case SHIFTL:
				{
					AST tmp92_AST = null;
					tmp92_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp92_AST);
					match(SHIFTL);
					break;
				}
				case SHIFTR:
				{
					AST tmp93_AST = null;
					tmp93_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp93_AST);
					match(SHIFTR);
					break;
				}
				default:
				{
					throw new NoViableAltException(LT(1), getFilename());
				}
				}
				}
				expr8();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop152;
			}
```

```
		} while (true);
		}
		expr7_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_22);
	}
	returnAST = expr7_AST;
}

public final void expr8() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST expr8_AST = null;

	try {	// for error handling
		expr9();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop156:
		do {
			if ((LA(1)==PLUS||LA(1)==MINUS)) {
				{
				switch ( LA(1)) {
				case PLUS:
				{
					AST tmp94_AST = null;
					tmp94_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp94_AST);
					match(PLUS);
					break;
				}
				case MINUS:
				{
					AST tmp95_AST = null;
					tmp95_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp95_AST);
					match(MINUS);
					break;
				}
				default:
				{
					throw new NoViableAltException(LT(1), getFilename());
				}
				}
				}
				expr9();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop156;
			}
```

```
		} while (true);
		}
		expr8_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_23);
	}
	returnAST = expr8_AST;
}

public final void expr9() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST expr9_AST = null;

	try {	// for error handling
		expr10();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop160:
		do {
			if (((LA(1) >= MULT && LA(1) <= MOD))) {
				{
				switch ( LA(1)) {
				case MULT:
				{
					AST tmp96_AST = null;
					tmp96_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp96_AST);
					match(MULT);
					break;
				}
				case DIV:
				{
					AST tmp97_AST = null;
					tmp97_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp97_AST);
					match(DIV);
					break;
				}
				case MOD:
				{
					AST tmp98_AST = null;
					tmp98_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp98_AST);
					match(MOD);
					break;
				}
				default:
				{
					throw new NoViableAltException(LT(1), getFilename());
```

```
                                }
                                }
                                }
                                expr10();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop160;
                        }

                } while (true);
                }
                expr9_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_24);
        }
        returnAST = expr9_AST;
}

public final void expr10() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expr10_AST = null;

        try {      // for error handling
                {
                {
                switch ( LA(1)) {
                case BW_NOT:
                {
                        AST tmp99_AST = null;
                        tmp99_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp99_AST);
                        match(BW_NOT);
                        break;
                }
                case MINUS:
                case NOT:
                case LPAREN:
                case REAL_T:
                case INTEGER_T:
                case STRING_T:
                case ID:
                case LITERAL_image:
                case LITERAL_pixel:
                case LITERAL_integer:
                case LITERAL_real:
                case LITERAL_string:
                case LITERAL_null:
                {
                        break;
```

```
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		expr11();
		astFactory.addASTChild(currentAST, returnAST);
		}
		expr10_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = expr10_AST;
}

public final void expr11() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST expr11_AST = null;

	try {	// for error handling
		{
		{
		switch ( LA(1)) {
		case NOT:
		{
			AST tmp100_AST = null;
			tmp100_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp100_AST);
			match(NOT);
			break;
		}
		case MINUS:
		case LPAREN:
		case REAL_T:
		case INTEGER_T:
		case STRING_T:
		case ID:
		case LITERAL_image:
		case LITERAL_pixel:
		case LITERAL_integer:
		case LITERAL_real:
		case LITERAL_string:
		case LITERAL_null:
		{
			break;
		}
		default:
		{
```

```
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                expr12();
                astFactory.addASTChild(currentAST, returnAST);
                }
                expr11_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                recover(ex,_tokenSet_11);
        }
        returnAST = expr11_AST;
}

public final void expr12() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expr12_AST = null;

        try {      // for error handling
                switch ( LA(1)) {
                case LITERAL_image:
                case LITERAL_pixel:
                case LITERAL_integer:
                case LITERAL_real:
                case LITERAL_string:
                case LITERAL_null:
                {
                        static_method();
                        astFactory.addASTChild(currentAST, returnAST);
                        expr12_AST = (AST)currentAST.root;
                        break;
                }
                case INTEGER_T:
                {
                        AST tmp101_AST = null;
                        tmp101_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp101_AST);
                        match(INTEGER_T);
                        expr12_AST = (AST)currentAST.root;
                        break;
                }
                case REAL_T:
                {
                        AST tmp102_AST = null;
                        tmp102_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp102_AST);
                        match(REAL_T);
                        expr12_AST = (AST)currentAST.root;
                        break;
                }
```

```
		case STRING_T:
		{
			AST tmp103_AST = null;
			tmp103_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp103_AST);
			match(STRING_T);
			expr12_AST = (AST)currentAST.root;
			break;
		}
		case MINUS:
		{
			AST tmp104_AST = null;
			tmp104_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp104_AST);
			match(MINUS);
			expr11();
			astFactory.addASTChild(currentAST, returnAST);
			expr12_AST = (AST)currentAST.root;
			break;
		}
		case LPAREN:
		{
			match(LPAREN);
			expr0();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			expr12_AST = (AST)currentAST.root;
			break;
		}
		default:
			if ((LA(1)==ID) && (LA(2)==LPAREN)) {
				function_call();
				astFactory.addASTChild(currentAST, returnAST);
				expr12_AST = (AST)currentAST.root;
			}
			else if ((LA(1)==ID) && (_tokenSet_25.member(LA(2)))) {
				lvalue();
				astFactory.addASTChild(currentAST, returnAST);
				expr12_AST = (AST)currentAST.root;
			}
		else {
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
	}
	catch (RecognitionException ex) {
		reportError(ex);
		recover(ex,_tokenSet_11);
	}
	returnAST = expr12_AST;
}

public static final String[] _tokenNames = {
```

"<0>",
"EOF",
"<2>",
"NULL_TREE_LOOKAHEAD",
"PLUS",
"MINUS",
"MULT",
"DIV",
"MOD",
"EQUALS",
"NOT_EQ",
"AND",
"OR",
"NOT",
"GT",
"LT",
"GTE",
"LTE",
"SHIFTL",
"SHIFTR",
"BW_AND",
"BW_OR",
"BW_NOT",
"BW_XOR",
"ASSIGN",
"SEMI",
"COLON",
"COMMA",
"DOT",
"STATIC",
"PROPERTY",
"LPAREN",
"RPAREN",
"LBRACE",
"RBRACE",
"LBRACKET",
"RBRACKET",
"UNDERSCORE",
"CHAR",
"DIGIT",
"SPACE",
"LF",
"CR",
"TAB",
"REAL_T",
"INTEGER_T",
"STRING_T",
"ID",
"NUMBER",
"STRING",
"WHITESPACE",
"COMMENT",
"VARS",
"PARAMS",

```
		"PARAM",
		"FBODY",
		"IFBODY",
		"\"program\"",
		"\"end\"",
		"\"image\"",
		"\"pixel\"",
		"\"integer\"",
		"\"real\"",
		"\"string\"",
		"\"null\"",
		"\"var\"",
		"\"red\"",
		"\"green\"",
		"\"blue\"",
		"\"width\"",
		"\"height\"",
		"\"create\"",
		"\"load\"",
		"\"save\"",
		"\"function\"",
		"\"main\"",
		"\"if\"",
		"\"else\"",
		"\"for\"",
		"\"while\"",
		"\"do\"",
		"\"return\"",
		"\"print\""
};

protected void buildTokenTypeASTClassMap() {
	tokenTypeToASTClassMap=null;
};

private static final long[] mk_tokenSet_0() {
	long[] data = { 2L, 0L};
	return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
	long[] data = { -576196841591267296L, 515075L, 0L, 0L};
	return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
private static final long[] mk_tokenSet_2() {
	long[] data = { 0L, 3072L, 0L, 0L};
	return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
private static final long[] mk_tokenSet_3() {
	long[] data = { 288230376151711744L, 0L};
	return data;
}
```

```
public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
private static final long[] mk_tokenSet_4() {
	long[] data = { 67108864L, 0L};
	return data;
}
public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());
private static final long[] mk_tokenSet_5() {
	long[] data = { 13589544960L, 0L};
	return data;
}
public static final BitSet _tokenSet_5 = new BitSet(mk_tokenSet_5());
private static final long[] mk_tokenSet_6() {
	long[] data = { 4429185024L, 0L};
	return data;
}
public static final BitSet _tokenSet_6 = new BitSet(mk_tokenSet_6());
private static final long[] mk_tokenSet_7() {
	long[] data = { 4294967296L, 0L};
	return data;
}
public static final BitSet _tokenSet_7 = new BitSet(mk_tokenSet_7());
private static final long[] mk_tokenSet_8() {
	long[] data = { -576196858771136480L, 512001L, 0L, 0L};
	return data;
}
public static final BitSet _tokenSet_8 = new BitSet(mk_tokenSet_8());
private static final long[] mk_tokenSet_9() {
	long[] data = { 17179869184L, 0L};
	return data;
}
public static final BitSet _tokenSet_9 = new BitSet(mk_tokenSet_9());
private static final long[] mk_tokenSet_10() {
	long[] data = { -576196841591267296L, 520193L, 0L, 0L};
	return data;
}
public static final BitSet _tokenSet_10 = new BitSet(mk_tokenSet_10());
private static final long[] mk_tokenSet_11() {
	long[] data = { 73211568112L, 0L};
	return data;
}
public static final BitSet _tokenSet_11 = new BitSet(mk_tokenSet_11());
private static final long[] mk_tokenSet_12() {
	long[] data = { 73182216192L, 0L};
	return data;
}
public static final BitSet _tokenSet_12 = new BitSet(mk_tokenSet_12());
private static final long[] mk_tokenSet_13() {
	long[] data = { 75359051760L, 0L};
	return data;
}
public static final BitSet _tokenSet_13 = new BitSet(mk_tokenSet_13());
private static final long[] mk_tokenSet_14() {
	long[] data = { -2879664302116044 96L, 523265L, 0L, 0L};
	return data;
```

```
}
public static final BitSet _tokenSet_14 = new BitSet(mk_tokenSet_14());
private static final long[] mk_tokenSet_15() {
	long[] data = { 73198993408L, 0L};
	return data;
}
public static final BitSet _tokenSet_15 = new BitSet(mk_tokenSet_15());
private static final long[] mk_tokenSet_16() {
	long[] data = { 73198997504L, 0L};
	return data;
}
public static final BitSet _tokenSet_16 = new BitSet(mk_tokenSet_16());
private static final long[] mk_tokenSet_17() {
	long[] data = { 73198999552L, 0L};
	return data;
}
public static final BitSet _tokenSet_17 = new BitSet(mk_tokenSet_17());
private static final long[] mk_tokenSet_18() {
	long[] data = { 73201096704L, 0L};
	return data;
}
public static final BitSet _tokenSet_18 = new BitSet(mk_tokenSet_18());
private static final long[] mk_tokenSet_19() {
	long[] data = { 73209485312L, 0L};
	return data;
}
public static final BitSet _tokenSet_19 = new BitSet(mk_tokenSet_19());
private static final long[] mk_tokenSet_20() {
	long[] data = { 73210533888L, 0L};
	return data;
}
public static final BitSet _tokenSet_20 = new BitSet(mk_tokenSet_20());
private static final long[] mk_tokenSet_21() {
	long[] data = { 73210535424L, 0L};
	return data;
}
public static final BitSet _tokenSet_21 = new BitSet(mk_tokenSet_21());
private static final long[] mk_tokenSet_22() {
	long[] data = { 73210781184L, 0L};
	return data;
}
public static final BitSet _tokenSet_22 = new BitSet(mk_tokenSet_22());
private static final long[] mk_tokenSet_23() {
	long[] data = { 73211567616L, 0L};
	return data;
}
public static final BitSet _tokenSet_23 = new BitSet(mk_tokenSet_23());
private static final long[] mk_tokenSet_24() {
	long[] data = { 73211567664L, 0L};
	return data;
}
public static final BitSet _tokenSet_24 = new BitSet(mk_tokenSet_24());
private static final long[] mk_tokenSet_25() {
	long[] data = { 107839741936L, 0L};
```

```
		return data;
	}
	public static final BitSet _tokenSet_25 = new BitSet(mk_tokenSet_25());

	}
```

FILE: IMLLexerTokenTypes.java

```
// $ANTLR 2.7.6 (20060127): "IML.g" -> "IMLLexer.java"$

public interface IMLLexerTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int PLUS = 4;
	int MINUS = 5;
	int MULT = 6;
	int DIV = 7;
	int MOD = 8;
	int EQUALS = 9;
	int AND = 10;
	int OR = 11;
	int NOT = 12;
	int BW_AND = 13;
	int BW_OR = 14;
	int BW_NOT = 15;
	int BW_XOR = 16;
	int ASSIGN = 17;
	int SEMI = 18;
	int LPAREN = 19;
	int RPAREN = 20;
	int UNDERSCORE = 21;
	int CHAR = 22;
	int DIGIT = 23;
	int SPACE = 24;
	int NL = 25;
	int LF = 26;
	int TAB = 27;
	int ID = 28;
	int NUMBER = 29;
	int STRING = 30;
	int WHITESPACE = 31;
}
```

FILE: IMLTokenTypes.java

```
// $ANTLR 2.7.6 (20060127): "IML.g" -> "IMLLexer.java"$

public interface IMLTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int PLUS = 4;
	int MINUS = 5;
	int MULT = 6;
	int DIV = 7;
```

```
int MOD = 8;
int EQUALS = 9;
int NOT_EQ = 10;
int AND = 11;
int OR = 12;
int NOT = 13;
int GT = 14;
int LT = 15;
int GTE = 16;
int LTE = 17;
int SHIFTL = 18;
int SHIFTR = 19;
int BW_AND = 20;
int BW_OR = 21;
int BW_NOT = 22;
int BW_XOR = 23;
int ASSIGN = 24;
int SEMI = 25;
int COLON = 26;
int COMMA = 27;
int DOT = 28;
int STATIC = 29;
int PROPERTY = 30;
int LPAREN = 31;
int RPAREN = 32;
int LBRACE = 33;
int RBRACE = 34;
int LBRACKET = 35;
int RBRACKET = 36;
int UNDERSCORE = 37;
int CHAR = 38;
int DIGIT = 39;
int SPACE = 40;
int LF = 41;
int CR = 42;
int TAB = 43;
int REAL_T = 44;
int INTEGER_T = 45;
int STRING_T = 46;
int ID = 47;
int NUMBER = 48;
int STRING = 49;
int WHITESPACE = 50;
int COMMENT = 51;
int VARS = 52;
int PARAMS = 53;
int PARAM = 54;
int FBODY = 55;
int IFBODY = 56;
int LITERAL_program = 57;
int LITERAL_end = 58;
int LITERAL_image = 59;
int LITERAL_pixel = 60;
int LITERAL_integer = 61;
```

```
	int LITERAL_real = 62;
	int LITERAL_string = 63;
	int LITERAL_null = 64;
	int LITERAL_var = 65;
	int LITERAL_red = 66;
	int LITERAL_green = 67;
	int LITERAL_blue = 68;
	int LITERAL_width = 69;
	int LITERAL_height = 70;
	int LITERAL_create = 71;
	int LITERAL_load = 72;
	int LITERAL_save = 73;
	int LITERAL_function = 74;
	int LITERAL_main = 75;
	int LITERAL_if = 76;
	int LITERAL_else = 77;
	int LITERAL_for = 78;
	int LITERAL_while = 79;
	int LITERAL_do = 80;
	int LITERAL_return = 81;
	int LITERAL_print = 82;
}
```

FILE: IMLType.java

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

interface IMLType  {
	void    setValue(Object value);
	Object getValue();
	int      getType();
}

class IMLNull implements IMLType, IMLTokenTypes {
	public void setValue(Object value){ }
	public Object getValue(){ return null; }
	public int getType(){ return 0; }
}

class IMLInteger extends IMLNull {
	private Integer m_value;
	public IMLInteger(){ m_value = new Integer(0); }
	public void setValue(Object value){ m_value = new Integer(((Number)value).intValue()); }
	public Object getValue(){ return m_value; }
	public int getType(){ return INTEGER_T; }
}

class IMLReal extends IMLNull {
	private Double m_value;
	public IMLReal(){ m_value = new Double(0.0); }
	public void setValue(Object value){ m_value = new Double(((Number)value).doubleValue()); }
```

```
	public Object getValue(){ return m_value; }
	public int getType(){ return REAL_T; }
}

class IMLString extends IMLNull {
	private String m_value;
	public IMLString(){ m_value = new String(); }
	public void setValue(Object value){ m_value = (String) value; }
	public Object getValue(){ return m_value; }
	public int getType(){ return STRING_T; }
}

class IMLPixel extends IMLNull {
	private Integer m_value;
	public IMLPixel(){ m_value = new Integer(0); }
	public void setValue(Object value){ m_value = new Integer(((Number)value).intValue()); }
	public Object getValue(){ return m_value; }
	public int getType(){ return LITERAL_pixel; }
}

class IMLImage extends IMLNull {
	private BufferedImage m_value;
	public IMLImage(){ m_value = new BufferedImage(160, 120,
BufferedImage.TYPE_INT_RGB); }
	public void setValue(Object value){ m_value = (BufferedImage) value; }
	public Object getValue(){ return m_value; }
	public int getType(){ return LITERAL_image; }
}
```

FILE: IMLVarFactory.java

```
class IMLVarFactory implements IMLTokenTypes {
	public static IMLType CreateVariable(int type){
		if(type == LITERAL_null) return new IMLNull();
		else if(type == LITERAL_integer || type == INTEGER_T) return new IMLInteger();
		else if(type == LITERAL_real || type == REAL_T) return new IMLReal();
		else if(type == LITERAL_string || type == STRING_T) return new IMLString();
		else if(type == LITERAL_pixel) return new IMLPixel();
		else if(type == LITERAL_image) return new IMLImage();
		else return null;
	}
}
```

FILE: IML.java

```
/**********************************************************************************************
	IML - Image Manipulation Language
	Written as a project for COMS 4115 (Programming Languages and Translators), Columbia
University
	Travis J. Galoppo (tjg2107@cs.columbia.edu)

	Many thanks to D.G. Yuengling and Sons, Bass & Co., and Arth Guinness, without whom this
project
	may actually have worked as I had originally intended!
```

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

import java.util.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;

class IML implements IMLTokenTypes {
	protected AST m_astProgram;
	protected String m_strProgramName;

	protected Hashtable m_globalVars;
	protected Hashtable m_globalFuncs;
	protected IMLFunction m_entryPoint;

	protected Stack	m_stack;
	protected Hashtablem_localVars;

	protected boolean	m_bVerbose;
	protected boolean	m_bOptimize;

	public static void main(String[] args){
		boolean bVerbose = false, bOptimize = false;
		String filename = null;

		if(args.length > 0){
			if(args[0].charAt(0) == '-'){
				for(int i=1; i<args[0].length(); i++){
					switch(args[0].charAt(i)){
					case 'o': bOptimize = true;  break;
					case 'v': bVerbose = true;   break;
					}
				}
				filename = args[1];
			} else filename = args[0];
		}

		if(filename != null){
			InputStream stream;
			try{ stream = new FileInputStream(filename);    }
			catch(FileNotFoundException e){
				System.out.println("File not found: " + filename);
				return;
			}
			new IML(bOptimize, bVerbose).go(stream);
		} else System.out.println("IML: no input file specified");
	}

	public IML(boolean bOptimize, boolean bVerbose){
		m_globalVars = new Hashtable();
```

```
        m_globalFuncs = new Hashtable();
        m_stack = new Stack();
        m_localVars = null;
        m_bVerbose = bVerbose;
        m_bOptimize = bOptimize;
}

protected void go(InputStream stream){
        try{
                IMLLexer lex = new IMLLexer(stream);
                IMLParser parser = new IMLParser(lex);
                parser.file();
                m_astProgram = parser.getAST();
                if(m_bVerbose){
                        System.out.println("UNOPTIMIZED TREE:");
                        DumpTree(m_astProgram, "");
                }
                Analyze();      // Check static semantics
                if(m_bOptimize){
                        Optimize();    // The optimizer does simple constant folding...
                        if(m_bVerbose){
                                System.out.println("OPTIMIZED TREE:");
                                DumpTree(m_astProgram, "");
                        }
                }
                Run();          // Run the program
        }catch(Exception e){
                System.out.println(e);
        }
}

protected void Analyze() throws Exception {
        if(m_astProgram != null){
                AST node = m_astProgram;
                if(node.getType() == LITERAL_program){
                        analyze_Program(node.getFirstChild());
                } else throw new Exception("Parser returned invalid AST");
        } else throw new Exception("No AST received from parser.");
}

protected void Optimize() throws Exception {
        for(Enumeration e = m_globalFuncs.keys(); e.hasMoreElements(); ){
                String fnName = (String)e.nextElement();
                if(fnName != null){
                        IMLFunction fn = (IMLFunction)m_globalFuncs.get(fnName);
                        optimize_FUNCTION(fn);
                }
        }
        optimize_FUNCTION(m_entryPoint);
}

protected void Run() throws Exception {
        run_FUNCTION(m_entryPoint, null);
}
```

```
/**********************************************************************************************
 * STATIC ANALYSIS FUNCTIONS -- Make sure everything makes sense and build symbol tables...
 **********************************************************************************************/

	protected void analyze_Program(AST node) throws Exception {
		if(node != null){
			m_strProgramName = node.getText();
			for(node = node.getNextSibling(); node != null; node = node.getNextSibling()){
				if(node.getType() == LITERAL_var) analyze_Vars(node, m_globalVars);
				else if(node.getType() == LITERAL_function) analyze_Function(node);
				else if(node.getType() == LITERAL_main) analyze_Main(node);
				else throw new Exception("Malformed AST; unknown token: '" +
node.getText() + "'");
			}
		} else throw new Exception("Malformed AST; program name not present");
	}

	protected void analyze_Vars(AST node, Hashtable context) throws Exception {
		if(node != null && node.getNumberOfChildren() == 2){
			AST varNames = node.getFirstChild();
			AST varType = varNames.getNextSibling();
			String varName;
			for(AST var=varNames.getFirstChild(); var != null; var=var.getNextSibling()){
				varName = var.getText();
				if(!context.containsKey(varName)){
					context.put(varName,
IMLVarFactory.CreateVariable(varType.getType()));
				} else throw new Exception("Redefinition of variable: " + varName);
			}
		} else throw new Exception("Malformed 'var' declaration");
	}

	protected void analyze_Function(AST node) throws Exception {
		if(node != null && node.getNumberOfChildren() == 4){
			AST fnName = node.getFirstChild();
			AST fnParams = fnName.getNextSibling();
			AST fnType = fnParams.getNextSibling();
			AST fnBody = fnType.getNextSibling();
			String funcName = fnName.getText();
			if(!m_globalFuncs.containsKey(funcName) &&
			  !m_globalVars.containsKey(funcName)){
				IMLFunction fn = new IMLFunction(fnParams, fnBody, fnType.getType());
				m_globalFuncs.put(funcName, fn);

				for(AST param = fnParams.getFirstChild(); param != null; param =
param.getNextSibling()){
					AST param_name = param.getFirstChild();
					AST param_type = param_name.getNextSibling();
					String paramName = param_name.getText();
					if(!fn.localVars.containsKey(paramName)){
						fn.localVars.put(paramName,
IMLVarFactory.CreateVariable(param_type.getType()));
					} else throw new Exception("Redefinition of variable '" +
```

```
paramName + "' in function " + funcName);
                                }

                                for(AST stmt = fnBody.getFirstChild(); stmt != null; stmt =
stmt.getNextSibling()){
                                        if(stmt.getType() == LITERAL_var) analyze_Vars(stmt,
fn.localVars);
                                        else analyze_Statement(stmt, fn.localVars);
                                }
                        } else throw new Exception("Redefinition of function: " + fnName);
                } else throw new Exception("Malformed 'function' declaration");
        }

        protected void analyze_Main(AST node) throws Exception {
                if(node != null){
                        AST body = node.getFirstChild();
                        m_entryPoint = new IMLFunction(null, body, LITERAL_null);
                        for(AST statement = body.getFirstChild(); statement != null; statement =
statement.getNextSibling()){
                                if(statement.getType() == LITERAL_var) analyze_Vars(statement,
m_entryPoint.localVars);
                                else analyze_Statement(statement, m_entryPoint.localVars);
                        }
                } else throw new Exception("Malformed MAIN declaration");
        }

        protected void analyze_Statement(AST node, Hashtable context) throws Exception{
                if(node != null){
                        switch(node.getType()){
                        case LITERAL_if: analyze_IF(node, context); break;
                        case LITERAL_for: analyze_FOR(node, context); break;
                        case LITERAL_while: analyze_WHILE(node, context); break;
                        case LITERAL_do: analyze_DO(node, context); break;
                        case LITERAL_return: analyze_RETURN(node, context); break;
                        case LITERAL_print: analyze_PRINT(node, context); break;
                        case LBRACE:
                                for(AST subNode=node.getFirstChild(); subNode!=null;
subNode=subNode.getNextSibling()){
                                        analyze_Statement(subNode, context);
                                }
                                break;
                        default: analyze_EXPRESSION(node, context);
                        }
                } else throw new Exception("Malformed function body");
        }

        protected void analyze_IF(AST node, Hashtable context) throws Exception {
                if(node != null && node.getNumberOfChildren() >= 2){
                        AST predicate = node.getFirstChild();
                        AST truePart = predicate.getNextSibling();
                        AST falsePart = truePart.getNextSibling();
                        analyze_Statement(predicate, context);
                        analyze_Statement(truePart, context);
                        if(falsePart != null) analyze_Statement(falsePart, context);
```

```
        } else throw new Exception("Malformed IF statement");
}

protected void analyze_FOR(AST node, Hashtable context) throws Exception {
        if(node != null){
                AST initPart = node.getFirstChild();
                AST predicatePart = initPart.getNextSibling();
                AST incrementPart = predicatePart.getNextSibling();
                AST body = incrementPart.getNextSibling();
                analyze_Statement(initPart, context);
                analyze_Statement(predicatePart, context);
                analyze_Statement(incrementPart, context);
                analyze_Statement(body, context);
        } else throw new Exception("Malformed FOR statement");
}

protected void analyze_WHILE(AST node, Hashtable context) throws Exception {
        if(node != null){
                AST predicate = node.getFirstChild();
                AST body = predicate.getNextSibling();
                analyze_Statement(predicate, context);
                analyze_Statement(body, context);
        } else throw new Exception("Malformed WHILE statement");
}

protected void analyze_DO(AST node, Hashtable context) throws Exception {
        if(node != null){
                AST body = node.getFirstChild();
                AST predicate = body.getNextSibling();
                analyze_Statement(body, context);
                analyze_Statement(predicate, context);
        } else throw new Exception("Malformed DO statement");
}

protected void analyze_RETURN(AST node, Hashtable context) throws Exception {
        if(node != null){
                analyze_EXPRESSION(node.getFirstChild(), context);
        } else throw new Exception("Malformed RETURN statement");
}

protected void analyze_PRINT(AST node, Hashtable context) throws Exception {
        if(node != null){
                for(AST expr = node.getFirstChild(); expr != null; expr = expr.getNextSibling()){
                        analyze_EXPRESSION(expr, context);
                }
        } else throw new Exception("Malformed PRINT statement");
}

protected int analyze_EXPRESSION(AST node, Hashtable context) throws Exception {
        int finalType = 0;
        switch(node.getType()){
        case PLUS: // plus is special because we can use it for string concatenation
                finalType = analyze_op_PLUS(node, context);
                break;
```

```
case MINUS:// minus is special becase it has a unary case
        finalType = analyze_op_MINUS(node, context);
        break;
case NOT:
case BW_NOT:
        finalType = analyze_op_UNARY(node, context);
        break;
case MULT:
case DIV:
case MOD:
case EQUALS:
case NOT_EQ:
case AND:
case OR:
case GT:
case LT:
case GTE:
case LTE:
case SHIFTL:
case SHIFTR:
case BW_AND:
case BW_OR:
case BW_XOR:
        finalType = analyze_op(node, context);
        break;
case ASSIGN:
        analyze_ASSIGN(node, context);
        finalType = 0;
        break;
case INTEGER_T:
case REAL_T:
case STRING_T:
case LITERAL_pixel:
case LITERAL_image:
        finalType = node.getType();
        break;
case ID:
        if(context.containsKey(node.getText())){
                finalType = getVarType(context, node.getText());
        } else if(m_globalVars.containsKey(node.getText())){
                finalType = getVarType(m_globalVars, node.getText());
        } else if(m_globalFuncs.containsKey(node.getText())){
                finalType = analyze_FUNCTION(node, context);
        } else throw new Exception("Unresolved symbol: " + node.getText());
        break;
case DOT:
        finalType = analyze_PROPERTY(node, context);
        break;
case STATIC:
        finalType = analyze_STATIC(node,context);
        break;
case LBRACKET:
        finalType = analyze_ARRAY(node, context);
        break;
```

```
		}
		return finalType;
	}

	protected int translate_Type(int oType){
		switch(oType){
		case LITERAL_string: return STRING_T;
		case LITERAL_real:   return REAL_T;
		case LITERAL_integer: return INTEGER_T;
		}
		return oType;
	}

	protected int analyze_PROPERTY(AST node, Hashtable context) throws Exception {
		AST id = node.getFirstChild();
		AST prop = id.getNextSibling();

		int nType;
		if(context.containsKey(id.getText())) nType = getVarType(context, id.getText());
		else if(m_globalVars.containsKey(id.getText())) nType = getVarType(m_globalVars,
id.getText());
		else throw new Exception("Unresolved symbol '" + id.getText() + "'");

		if(nType == LITERAL_pixel) return LITERAL_pixel;     // color components are
themselves pixels
		else if(nType == LITERAL_image) return INTEGER_T;         // width/height are
integers.
		else throw new Exception("Variable '" + id.getText() + "' does not have a property '" +
prop.getText() + "'");
	}

	protected int analyze_STATIC(AST node, Hashtable context) throws Exception {
		AST type = node.getFirstChild();
		if(type.getType() == LITERAL_image){
			AST method = type.getNextSibling();
			switch(method.getType()){
			case LITERAL_create:
			case LITERAL_load:         return LITERAL_image;
			case LITERAL_save:         return 0;
			default: throw new Exception("Type image has no static method '" +
method.getText() + "'");
			}
		} else throw new Exception("Type '" + type.getType() + "' has no static methods");
	}

	protected int analyze_ARRAY(AST node, Hashtable context) throws Exception {
		if(node.getNumberOfChildren() == 3){
			AST id = node.getFirstChild();
			AST x = id.getNextSibling();
			AST y = x.getNextSibling();

			int id_type;
			if(context.containsKey(id.getText())) id_type = getVarType(context, id.getText());
			else if(m_globalVars.containsKey(id.getText())) id_type =
```

```
getVarType(m_globalVars, id.getText());
                        else throw new Exception("Unresolved symbol: '" + id.getText() + "'");

                        if(id_type == LITERAL_image){
                                int x_type = analyze_EXPRESSION(x, context);
                                int y_type = analyze_EXPRESSION(y, context);
                                if(x_type == y_type && y_type == INTEGER_T) return LITERAL_pixel;
                        } else throw new Exception("Array index only valid for IMAGE objects");
                } else throw new Exception("Pixel index must be x,y");
                return 0;
        }

        protected int analyze_FUNCTION(AST node, Hashtable context) throws Exception {
                IMLFunction fn = (IMLFunction) m_globalFuncs.get(node.getText());
                if(fn.params.getNumberOfChildren() != node.getNumberOfChildren()){
                        throw new Exception("Call to function '" + node.getText() + "' with improper
number of arguments");
                }
                AST param = node.getFirstChild();
                AST fParam = fn.params.getFirstChild();
                while(param != null){
                        int pType = analyze_EXPRESSION(param, context);
                        int fType = translate_Type(fParam.getFirstChild().getNextSibling().getType());
                        if(!analyze_COMPATIBLETYPES(pType, fType)){
                                throw new Exception("Call to function '" + node.getText() + "' with
improper arg type");
                        }
                        param = param.getNextSibling();
                        fParam = fParam.getNextSibling();
                }
                return fn.type;
        }

        protected int analyze_op_PLUS(AST node, Hashtable context) throws Exception {
                AST operand1 = node.getFirstChild();
                AST operand2 = operand1.getNextSibling();
                int op1_type = analyze_EXPRESSION(operand1, context);
                int op2_type = analyze_EXPRESSION(operand2, context);
                int finalType=0;
                if(op1_type == op2_type && op2_type == STRING_T){
                        finalType = STRING_T;
                } else  finalType = analyze_RETURNTYPE(op1_type, op2_type);
                return finalType;
        }

        protected int analyze_op_MINUS(AST node, Hashtable context) throws Exception {
                AST op1 = node.getFirstChild();
                AST op2 = op1.getNextSibling();
                if(op2 != null) return analyze_op(node, context);
                else return analyze_op_UNARY(node, context);
        }

        protected int analyze_op_UNARY(AST node, Hashtable context) throws Exception {
                AST op = node.getFirstChild();
```

```
        return analyze_EXPRESSION(op, context);
}

protected int analyze_op(AST node, Hashtable context) throws Exception {
        AST operand1 = node.getFirstChild();
        AST operand2 = operand1.getNextSibling();
        int op1_type = analyze_EXPRESSION(operand1, context);
        int op2_type = analyze_EXPRESSION(operand2, context);
        int finalType=0;
        if( (op1_type == STRING_T || op2_type == STRING_T) ){
                throw new Exception("Operator " + node.getText() + " : invalid operand types");
        } else  finalType = analyze_RETURNTYPE(op1_type, op2_type);
        return finalType;
}

protected void analyze_ASSIGN(AST node, Hashtable context) throws Exception {
        AST lvalue = node.getFirstChild();
        AST rvalue = lvalue.getNextSibling();
                // make sure lvalue is not a constant or a function
        int lvalue_type = lvalue.getType();
        if(lvalue_type == ID || lvalue_type == DOT || lvalue_type == LBRACKET ){
                if(m_globalFuncs.containsKey(node.getText())){
                        throw new Exception("Can not assign to function name");
                }
                if(!analyze_COMPATIBLETYPES(analyze_EXPRESSION(lvalue, context),
                                        analyze_EXPRESSION(rvalue, context))){
                        throw new Exception("Operator = : incompatible operand types");
                }
        } else throw new Exception("Operator = : lvalue must be a variable");
}

protected int getVarType(Hashtable table, String name){
        int type = 0;
        if(table.containsKey(name)){
                type = ((IMLType)table.get(name)).getType();
        }
        return type;
}

protected int analyze_RETURNTYPE(int op1_type, int op2_type) throws Exception {
        if( op1_type == op2_type ) return op1_type;
        else if(op1_type == STRING_T || op2_type == STRING_T) return STRING_T;
        else if(op1_type == INTEGER_T && op2_type == REAL_T) return REAL_T;
        else if(op1_type == REAL_T && op2_type == INTEGER_T) return REAL_T;
        else if(op1_type == INTEGER_T && op2_type == LITERAL_pixel) return INTEGER_T;
        else if(op1_type == LITERAL_pixel && op2_type == INTEGER_T) return INTEGER_T;
        else if(op1_type == LITERAL_pixel && op2_type == REAL_T) return REAL_T;
        else if(op1_type == REAL_T && op2_type == LITERAL_pixel) return REAL_T;
        else throw new Exception("Illegal operation; mismatched operands");
}

protected boolean analyze_COMPATIBLETYPES(int type1, int type2) throws Exception {
        if(type1 == STRING_T || type2 == STRING_T ||
          type1 == LITERAL_image || type2 == LITERAL_image){
```

```
                return type1 == type2;
            } else return true;
        }

/*********************************************************************************************
 * OPTIMIZER FUNCTIONS -- Performs constant folding...
 *********************************************************************************************/

        protected void optimize_FUNCTION(IMLFunction fn) throws Exception {
            optimize_ASTNode(fn.body.getFirstChild());
        }

        protected void optimize_ASTNode(AST node) throws Exception {
            while(node != null){
                optimize_STATEMENT(node);
                node = node.getNextSibling();
            }
        }

        protected void optimize_STATEMENT(AST node) throws Exception {
            if(node != null){
                switch(node.getType()){
                case ID:
                case STATIC:                optimize_EXPRESSION(node);
                case ASSIGN:                optimize_ASSIGN(node); break;
                case LITERAL_if:    optimize_IF(node); break;
                case LITERAL_for:  optimize_FOR(node); break;
                case LITERAL_while:        optimize_WHILE(node); break;
                case LITERAL_do:  optimize_DO(node); break;
                case LITERAL_print:        optimize_PRINT(node); break;
                case LBRACE:                optimize_ASTNode(node.getFirstChild()); break;
                case LITERAL_return:       optimize_EXPRESSION(node.getFirstChild());
                }
            }
        }

        protected void optimize_ASSIGN(AST node) throws Exception {
            AST lvalue = node.getFirstChild();
            AST expr = lvalue.getNextSibling();
            optimize_EXPRESSION(expr);
        }

        protected void optimize_IF(AST node) throws Exception {
            AST pred = node.getFirstChild();
            AST truepart = pred.getNextSibling();
            AST falsepart = truepart.getNextSibling();
            optimize_EXPRESSION(pred);
            optimize_STATEMENT(truepart);
            optimize_STATEMENT(falsepart);
        }

        protected void optimize_FOR(AST node) throws Exception {
            AST init = node.getFirstChild();
            AST pred = init.getNextSibling();
```

```
        AST incr = pred.getNextSibling();
        AST stmt = incr.getNextSibling();
        optimize_EXPRESSION(init);
        optimize_EXPRESSION(pred);
        optimize_EXPRESSION(incr);
        optimize_STATEMENT(stmt);
}

protected void optimize_WHILE(AST node) throws Exception {
        AST pred = node.getFirstChild();
        AST stmt = pred.getNextSibling();
        optimize_EXPRESSION(pred);
        optimize_STATEMENT(stmt);
}

protected void optimize_DO(AST node) throws Exception {
        AST stmt = node.getFirstChild();
        AST pred = stmt.getNextSibling();
        optimize_EXPRESSION(pred);
        optimize_STATEMENT(stmt);
}

protected void optimize_PRINT(AST node) throws Exception {
        node = node.getFirstChild();
        while(node != null){
                optimize_EXPRESSION(node);
                node = node.getNextSibling();
        }
}

protected int optimize_EXPRESSION(AST node) throws Exception {
        if(node != null){
                AST L, R;
                int a=0, b=0;
                switch(node.getType()){
                        case STRING_T:
                        case REAL_T:
                        case INTEGER_T:
                                return 1;
                        case PLUS:
                        case MULT:
                        case DIV:
                        case MOD:
                        case SHIFTL:
                        case SHIFTR:
                        case BW_AND:
                        case BW_OR:
                        case BW_XOR:
                        case EQUALS:
                        case NOT_EQ:
                        case AND:
                        case OR:
                        case GT:
                        case GTE:
```

```
                        case LT:
                        case LTE:
                                L = node.getFirstChild();
                                R = L.getNextSibling();
                                a = optimize_EXPRESSION(L);
                                b = optimize_EXPRESSION(R);
                                if(a + b == 2){
                                        Object val = run_EXPRESSION(node);
                                        node.setFirstChild(null);
                                        node.setText(val.toString());
                                        if(val instanceof String) node.setType(STRING_T);
                                        else if(val instanceof Double) node.setType(REAL_T);
                                        else node.setType(INTEGER_T);
                                        return 1;
                                }
                                break;
                        }
                        return 0;
                } else return 0;
        }

/*****************************************************************************************
 * EXECUTION FUNCTIONS
 *****************************************************************************************/

        protected Object run_FUNCTION(IMLFunction fn, AST parameters) throws Exception {
                Object rVal = null;

                        // establish "stack frame"
                Hashtable locals = new Hashtable();
                for(Enumeration e = fn.localVars.keys(); e.hasMoreElements(); ){
                        String sName = (String)e.nextElement();
                        if(sName != null){
                                locals.put(sName,
IMLVarFactory.CreateVariable(((IMLType)fn.localVars.get(sName)).getType()));
                        }
                }

                        // fill in parameters...
                if(parameters != null){
                        AST param = parameters;
                        AST fParam = fn.params.getFirstChild();
                        while(fParam != null){
                                String pName = fParam.getFirstChild().getText();
                                ((IMLType)locals.get(pName)).setValue( run_EXPRESSION(param) );
                                param = param.getNextSibling();
                                fParam = fParam.getNextSibling();
                        }
                }

                m_stack.push(m_localVars);
                m_localVars = locals;
                rVal = run_ASTNode(fn.body.getFirstChild());
                m_localVars = (Hashtable) m_stack.pop();
```

```
        return rVal;
}

protected Object run_ASTNode(AST node) throws Exception {
        Object rVal = null;
        while(node != null){
                rVal = run_STATEMENT(node);
                node = node.getNextSibling();
        }
        return rVal;
}

protected Object run_STATEMENT(AST node) throws Exception {
        if(node != null){
                switch(node.getType()){
                case ID:
                case STATIC: return run_EXPRESSION(node);
                case ASSIGN: run_ASSIGN(node); break;
                case LITERAL_if: run_IF(node); break;
                case LITERAL_for: run_FOR(node); break;
                case LITERAL_while: run_WHILE(node); break;
                case LITERAL_do: run_DO(node); break;
                case LITERAL_print: run_PRINT(node); break;
                case LBRACE: run_ASTNode(node.getFirstChild()); break;
                case LITERAL_return: return run_EXPRESSION(node.getFirstChild());
                }
        }
        return null;
}

protected Object run_IF(AST node) throws Exception {
        AST pred = node.getFirstChild();
        AST truepart = pred.getNextSibling();
        AST falsepart = truepart.getNextSibling();

        Object val = run_EXPRESSION(pred);
        int iVal = ((Number)val).intValue();
        if(iVal != 0) return run_STATEMENT(truepart);
        else if(falsepart != null) return run_STATEMENT(falsepart);
        return null;
}

protected Object run_FOR(AST node) throws Exception {
        AST init = node.getFirstChild();
        AST pred = init.getNextSibling();
        AST incr = pred.getNextSibling();
        AST stmt = incr.getNextSibling();

        run_STATEMENT(init);
        Object pVal = run_EXPRESSION(pred);
        int iVal = ((Number)pVal).intValue();
        while(iVal != 0){
                run_STATEMENT(stmt);
                run_STATEMENT(incr);
```

```
                pVal = run_EXPRESSION(pred);
                iVal = ((Number)pVal).intValue();
        }
        return null;
    }

    protected Object run_WHILE(AST node) throws Exception {
        AST pred = node.getFirstChild();
        AST stmt = pred.getNextSibling();

        Object pVal = run_EXPRESSION(pred);
        int iVal = ((Integer)pVal).intValue();
        while(iVal != 0){
                run_STATEMENT(stmt);
                pVal = run_EXPRESSION(pred);
                iVal = ((Integer)pVal).intValue();
        }
        return null;
    }

    protected Object run_DO(AST node) throws Exception {
        AST stmt = node.getFirstChild();
        AST pred = stmt.getNextSibling();
        Object pVal;
        int iVal=0;

        do{
                run_STATEMENT(stmt);
                pVal = run_EXPRESSION(pred);
                iVal = ((Number)pVal).intValue();
        } while(iVal != 0);
        return null;
    }

    protected IMLType resolve_VAR(String varName){
        if(m_localVars.containsKey(varName)) return (IMLType) m_localVars.get(varName);
        else if(m_globalVars.containsKey(varName)) return (IMLType)
m_globalVars.get(varName);
        else return null;
    }

    protected void run_ASSIGN(AST node) throws Exception {
        AST lvalue = node.getFirstChild();
        AST expr = lvalue.getNextSibling();

        Object val = run_EXPRESSION(expr);
        if(lvalue.getType() == ID){
                IMLType var=resolve_VAR(lvalue.getText());
                if(var != null) var.setValue(val);
        } else if(lvalue.getType() == DOT){
                run_SET_PROPERTY(lvalue, val);
        } else if(lvalue.getType() == LBRACKET){
                run_SET_ARRAYINDEX(lvalue, val);
        }
```

```
}

protected void run_SET_PROPERTY(AST node, Object val) throws Exception {
	AST id = node.getFirstChild();
	AST prop = id.getNextSibling();
	int v = ((Number)val).intValue();
	IMLType var = resolve_VAR(id.getText());
	if(var != null){
		if(var.getType() == LITERAL_pixel){
			if(v > 255) throw new Exception("Value too high for pixel component");
			int mask=0;
			switch(prop.getType()){
			case LITERAL_red: v <<= 16; mask = 0x00FF0000; break;
			case LITERAL_green: v <<= 8; mask = 0x0000FF00; break;
			case LITERAL_blue: mask = 0x000000FF; break;
			}
			int p = ((Number)var.getValue()).intValue();
			p = (p & ~mask) | v;
			var.setValue(new Integer(p));
		} else throw new Exception("Property '" + prop.getText() + "' is read only!");
	}
}

protected void run_SET_ARRAYINDEX(AST node, Object val) throws Exception {
	AST id = node.getFirstChild();
	AST x = id.getNextSibling();
	AST y = x.getNextSibling();
	int v = ((Number)val).intValue();
	IMLType var = resolve_VAR(id.getText());
	if(var != null){
		if(var.getType() == LITERAL_image){
			BufferedImage img = (BufferedImage)var.getValue();
			Object x_val = run_EXPRESSION(x);
			Object y_val = run_EXPRESSION(y);
			int X = ((Number)x_val).intValue();
			int Y = ((Number)y_val).intValue();
			img.setRGB(X, Y, v);
		}
	}
}

protected void run_PRINT(AST node) throws Exception {
	node = node.getFirstChild();
	while(node != null){
		System.out.print(run_EXPRESSION(node));
		node = node.getNextSibling();
	}
	System.out.println();
}

protected Object run_EXPRESSION(AST node) throws Exception {
	AST a, b;
	switch(node.getType()){
	case STRING_T:	return node.getText();
```

```
case INTEGER_T:
case LITERAL_pixel:
		return new Integer( Integer.parseInt( node.getText() ) );
case REAL_T:	return new Double( Double.parseDouble( node.getText() ) );
case ID:
	if(m_localVars.containsKey(node.getText())){
		return ((IMLType)m_localVars.get(node.getText())).getValue();
	} else if(m_globalVars.containsKey(node.getText())){
		return ((IMLType)m_globalVars.get(node.getText())).getValue();
	} else if(m_globalFuncs.containsKey(node.getText())){
		IMLFunction fn = (IMLFunction)m_globalFuncs.get(node.getText());
		return run_FUNCTION(fn, node.getFirstChild());
	}
	break;
case DOT:	return run_GET_PROPERTY(node);
case LBRACKET:	return run_GET_ARRAYINDEX(node);
case STATIC:	return run_STATICMETHOD(node);
case PLUS:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _add(run_EXPRESSION(a), run_EXPRESSION(b));
case MINUS:
	a = node.getFirstChild(); b = a.getNextSibling();
	if(b == null) return _neg(run_EXPRESSION(a));
	else return _sub(run_EXPRESSION(a), run_EXPRESSION(b));
case MULT:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _mul(run_EXPRESSION(a), run_EXPRESSION(b));
case DIV:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _div(run_EXPRESSION(a), run_EXPRESSION(b));
case MOD:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _mod(run_EXPRESSION(a), run_EXPRESSION(b));
case EQUALS:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _eq(run_EXPRESSION(a), run_EXPRESSION(b));
case NOT_EQ:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _ne(run_EXPRESSION(a), run_EXPRESSION(b));
case GT:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _gt(run_EXPRESSION(a), run_EXPRESSION(b));
case LT:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _lt(run_EXPRESSION(a), run_EXPRESSION(b));
case GTE:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _gte(run_EXPRESSION(a), run_EXPRESSION(b));
case LTE:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _lte(run_EXPRESSION(a), run_EXPRESSION(b));
case AND:
	a = node.getFirstChild(); b = a.getNextSibling();
	return _and(a,b);
```

```
		case OR:
			a = node.getFirstChild(); b = a.getNextSibling();
			return _or(a,b);
		case NOT:
			a = node.getFirstChild();
			return _not(run_EXPRESSION(a));
		case SHIFTL:
			a = node.getFirstChild(); b = a.getNextSibling();
			return _bwshl(run_EXPRESSION(a), run_EXPRESSION(b));
		case SHIFTR:
			a = node.getFirstChild(); b = a.getNextSibling();
			return _bwshr(run_EXPRESSION(a), run_EXPRESSION(b));
		case BW_AND:
			a = node.getFirstChild(); b = a.getNextSibling();
			return _bwand(run_EXPRESSION(a), run_EXPRESSION(b));
		case BW_OR:
			a = node.getFirstChild(); b = a.getNextSibling();
			return _bwor(run_EXPRESSION(a), run_EXPRESSION(b));
		case BW_XOR:
			a = node.getFirstChild(); b = a.getNextSibling();
			return _bwxor(run_EXPRESSION(a), run_EXPRESSION(b));
		case BW_NOT:
			a = node.getFirstChild();
			return _bwnot(run_EXPRESSION(a));
		}
		return null;
	}

	protected Object run_GET_PROPERTY(AST node){
		AST id = node.getFirstChild();
		AST prop = id.getNextSibling();

		IMLType var=resolve_VAR(id.getText());
		if(var != null){
			if(var.getType() == LITERAL_pixel){
				int p = ((Number)var.getValue()).intValue();
				switch(prop.getType()){
				case LITERAL_red: p>>=16; break;
				case LITERAL_green: p>>=8; break;
				}
				return new Integer(p & 0x000000FF);
			} else {
				BufferedImage img = (BufferedImage)var.getValue();
				if(prop.getType() == LITERAL_width) return new Integer(img.getWidth());
				else if(prop.getType() == LITERAL_height) return new
Integer(img.getHeight());
			}
		}
		return null;
	}

	protected Object run_GET_ARRAYINDEX(AST node) throws Exception {
		AST id = node.getFirstChild();
		AST x = id.getNextSibling();
```

```
        AST y = x.getNextSibling();
        IMLType var=resolve_VAR(id.getText());
        if(var != null){
                if(var.getType() == LITERAL_image){
                        BufferedImage img = (BufferedImage)var.getValue();
                        Object x_val = run_EXPRESSION(x);
                        Object y_val = run_EXPRESSION(y);
                        int X = ((Number)x_val).intValue();
                        int Y = ((Number)y_val).intValue();
                        return new Integer(img.getRGB(X,Y) & 0x00FFFFFF);
                }
        }
        return null;
}

protected Object run_STATICMETHOD(AST node) throws Exception {
        AST type = node.getFirstChild(); // we know this is image, from analysis
        AST method = type.getNextSibling();
        switch(method.getType()){
        case LITERAL_create: return run_CREATEIMAGE(method);
        case LITERAL_load:   return run_LOADIMAGE(method);
        case LITERAL_save:   return run_SAVEIMAGE(method);
        }
        return null;
}

protected Object run_CREATEIMAGE(AST node) throws Exception {
        AST width = node.getNextSibling();
        AST height = width.getNextSibling();
        Object ow = run_EXPRESSION(width);
        Object oh = run_EXPRESSION(height);
        int w = ((Number)ow).intValue();
        int h = ((Number)oh).intValue();
        return new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
}

protected Object run_LOADIMAGE(AST node) throws Exception {
        AST filename = node.getNextSibling();
        Object of = run_EXPRESSION(filename);
        BufferedImage bi = (BufferedImage) ImageIO.read(new File((String)of));
        return bi;
}

protected Object run_SAVEIMAGE(AST node) throws Exception {
        AST img = node.getNextSibling();
        AST filename = img.getNextSibling();
        IMLType var=resolve_VAR(img.getText());
        if(var != null){
                Object of = run_EXPRESSION(filename);
                ImageIO.write((RenderedImage)var.getValue(), "jpg", new File((String)of));
        }
        return null;
}
```

```
protected Object _add(Object a, Object b){
	if(a instanceof String || b instanceof String) return a.toString() + b.toString();
	else {
		Number A = (Number)a, B = (Number)b;
		if(a instanceof Integer && b instanceof Integer)
			return new Integer(A.intValue() + B.intValue());
		else return new Double(A.doubleValue() + B.doubleValue());
	}

}

protected Object _sub(Object a, Object b){
	Number A = (Number)a, B = (Number) b;
	if(a instanceof Integer && b instanceof Integer)
		return new Integer(A.intValue() - B.intValue());
	else return new Double(A.doubleValue() - B.doubleValue());
}

protected Object _mul(Object a, Object b){
	Number A = (Number)a, B = (Number) b;
	if(a instanceof Integer && b instanceof Integer)
		return new Integer(A.intValue() * B.intValue());
	else return new Double(A.doubleValue() * B.doubleValue());
}

protected Object _div(Object a, Object b){
	Number A = (Number)a, B = (Number) b;
	if(a instanceof Integer && b instanceof Integer)
		return new Integer(A.intValue() / B.intValue());
	else return new Double(A.doubleValue() / B.doubleValue());
}

protected Object _neg(Object a){
	if(a instanceof Integer){
		return new Integer(-((Integer)a).intValue());
	} else return new Double(-((Double)a).doubleValue());
}

protected Object _mod(Object a, Object b){
	int _a = ((Number)a).intValue();
	int _b = ((Number)b).intValue();
	return new Integer(_a % _b);
}

protected Object _eq(Object a, Object b){
	int rval;
	if( (a instanceof Double) || (b instanceof Double) ){
		rval = ((Number)a).doubleValue() == ((Number)b).doubleValue() ? 1 : 0;
	} else rval = ((Number)a).intValue() == ((Number)b).intValue() ? 1 : 0;
	return new Integer(rval);
}

protected Object _ne(Object a, Object b){
	return new Integer(1 - ((Number)_eq(a,b)).intValue());
```

```
}

protected Object _gt(Object a, Object b){
	int rval;
	if( (a instanceof Double) || (b instanceof Double) ){
		rval = ((Number)a).doubleValue() > ((Number)b).doubleValue() ? 1 : 0;
	} else  rval = ((Number)a).intValue() > ((Number)b).intValue() ? 1 : 0;
	return new Integer(rval);
}

protected Object _lt(Object a, Object b){
	int rval;
	if( (a instanceof Double) || (b instanceof Double) ){
		rval = ((Number)a).doubleValue() < ((Number)b).doubleValue() ? 1 : 0;
	} else  rval = ((Number)a).intValue() < ((Number)b).intValue() ? 1 : 0;
	return new Integer(rval);
}

protected Object _gte(Object a, Object b){
	return new Integer(1 - ((Number)_lt(a,b)).intValue());
}

protected Object _lte(Object a, Object b){
	return new Integer(1 - ((Number)_gt(a,b)).intValue());
}

	// lazy AND
protected Object _and(AST a, AST b) throws Exception {
	int _a = ((Number)run_EXPRESSION(a)).intValue();
	if(_a != 0){
		int _b = ((Number)run_EXPRESSION(b)).intValue();
		if(_b != 0) return 1;
	}
	return 0;
}

	// lazy OR
protected Object _or(AST a, AST b) throws Exception {
	int _a = ((Number)run_EXPRESSION(a)).intValue();
	if(_a != 0) return 1;
	int _b = ((Number)run_EXPRESSION(b)).intValue();
	if(_b != 0) return 1;
	return 0;
}

protected Object _not(Object a){
	return new Integer( (((Number)a).intValue() > 0) ? 0 : 1 );
}

protected Object _bwshl(Object a, Object b){
	int A = ((Number)a).intValue();
	int B = ((Number)b).intValue();
	return new Integer(A << B);
}
```

```
	protected Object _bwshr(Object a, Object b){
		int A = ((Number)a).intValue();
		int B = ((Number)b).intValue();
		return new Integer(A >> B);
	}

	protected Object _bwand(Object a, Object b){
		int A = ((Number)a).intValue();
		int B = ((Number)b).intValue();
		return new Integer(A & B);
	}

	protected Object _bwor(Object a, Object b){
		int A = ((Number)a).intValue();
		int B = ((Number)b).intValue();
		return new Integer(A | B);
	}

	protected Object _bwxor(Object a, Object b){
		int A = ((Number)a).intValue();
		int B = ((Number)b).intValue();
		return new Integer(A ^ B);
	}

	protected Object _bwnot(Object a){
		int A = ((Number)a).intValue();
		return new Integer(~A);
	}

		// This is a debug function to dump the tree...
	private void DumpTree(AST root, String spc){
		if(root != null){
			System.out.println(spc + "Node [" + root.getText() + "] {" + root.getType() + "}");
			DumpTree(root.getFirstChild(), spc+"  ");
			DumpTree(root.getNextSibling(), spc);
		}
	}
}

class IMLFunction implements IMLTokenTypes {
	public AST params;
	public AST body;
	public int type;
	public Hashtable localVars;
	public IMLFunction(AST paramlist, AST fnbody, int returnType){
		params = paramlist;
		body = fnbody;
		if(returnType == LITERAL_null) type = 0;
		else if(returnType == LITERAL_integer) type = INTEGER_T;
		else if(returnType == LITERAL_real) type = REAL_T;
		else if(returnType == LITERAL_string) type = STRING_T;
		else type = returnType;
		localVars = new Hashtable();
```

```
        }
}
```