COMS W4115: Programming Languages and Translators

MATLIP: MATLAB-Like Language for Image Processing

Final Report

Pin-Chin Huang (ph2249@columbia.edu)
Shariar Zaber Kazi (szk2103@columbia.edu)
Shih-Hao Liao (sl2937@columbia.edu)
PoHsu Yeh (py2157@columbia.edu)

December 19, 2008

1. INTRODUCTION

Today, there are numerous image processing applications available such as Adobe Photoshop, Picture-it, Picassa, etc. However, these applications do not provide any programmatic ability to process images, which is required by image processing systems such as iris pattern recognition system. These image processing centric systems require conversion of image colors, image rotation, blurring, sharpening, resizing, edge detection and a lot of other processing. There are a number of languages available which offer such programmatic image processing, such as C++, Java, MATLAB, etc. Of these, only MATLAB provides ease of programming images for both novice and advanced users. However, because of the high cost in purchasing the license for MATLAB, it often deters many users in buying. Here, we propose a simple MATLAB like language syntax for simple and easy image processing. We call it MATLIP (***Mat**lab **L**ike **I**mage **P**rocessing*).

Since, our language will provide image manipulation, we start by a simple description of how images are represented in computers. A modern computer image, at the point where it is presented *(rendered)* for human consumption, usually consists of a rectangular array of closely spaced colored dots. Ideally, these dots are so small and so close together that the human eye cannot distinguish them individually. This causes them to run together and appear to represent continuous color. The individual dots are commonly referred to as pixels, which are derived from the term picture elements.

The pixels are typically stored and transported in files, and are then extracted from the files and displayed on a computer screen or a sheet of paper for human consumption. There are a fairly large number of formats for storing the pixels in a file. Different file formats have advantages and disadvantages in terms of compression, size, reproduction quality, etc. Our language will support reading standard image file formats such as JPEG, TIFF, BMP, GIF etc. by using our simple built-in function `imread()`. The language support provides manipulation of a single pixel, or a group of pixels, or an entire image. For the image we have a built-in type "`Image`". Since, often image processing requires a 2-D matrix which is applied over the image for various algorithms such as convolution, edge detection etc., we also have another built-in type "`Kernel`" to provide such functionalities.

2. LANGUAGE TUTORIAL

2.1. Variable declaration

There are five kinds of variable declaration in our language, int, float, boolean, kernel, and image, all of which should be declared at the beginning of the program or function. For example,

```
int x;
function = main()
int y;
end
```

GOOD

```
function = main()
...not declaration…..
int x;(not ok)
end
int y; (not ok)
```

WRONG

2.2. Variable assignment

Variable should be declared before being assigned a value. Besides, declaration and assignment cannot be done at the same time and their type should match with each other.

```
int x;
function = main()
int y;
y=3;
x=3;
end
```

GOOD

```
int x=3;(not ok)
function = main()
int y;
y=3.0;(not ok)
end
```

WRONG

2.3. Arithmetic Operation

Our language supports +,-,*,/,^(power), and mod. For the variable, int and float, +,-,*,/,^(power), and mod can be applied. For the variable image and kernel, only +,-,*, and / are allowed. Besides, both sides of the operator should be of the same type in most of the cases except image and kernel. In order to +,-,* and / a constant for each element of image and kernel, the syntax - image variable operator int variable and image variable operator float variable is permitted. The following are the examples for the arithmetic operation.

```
image x;
image y;
kernel k1;
function = main()
x=x*2;
x=x+y;
k1=k1*2.0;
end
```

GOOD

```
image x;
kernel k;
function = main()
x=2*x;(not ok)
x=x*2.0;(not ok)
x=x+k;(not ok)
end
```

WRONG

2.4.Control Flow Statement

In our language, if…else if ….else, while and for control flow statement are supported. "If" statement is used to do a certain action when condition is met while "when" and "for" is used to do an action many times when condition is met.

The following is the syntax for the "if" statement. Note that the condition placed after if or else if should be of bool type.

```
function = main()
int x;
if x==0
  x=x+1;
elseif x==1
  x=x+2;
elseif x==3
  x=x+3;
else
  x=x+4;
end
end
```

GOOD

```
function = main()
int x;
if x==0
  x=x+1;
end
if x==1
  x=x+1;
else
  x=x+2;
end
end
```

GOOD

The following is the syntax for the "for" and "while" statement. The condition placed after if or else if should be of bool type. Besides, for i=0:1:10 means that I is from 0 to 10 and each time i will be incremented by one.

```
function = main()
int x;
x=3;
while x>0
  x=x-1;
end
end
```

GOOD

```
function = main()
int i;
for i=0:1:10
    i=i+1;
end
end
```

GOOD

2.5. Function

Function should be declared before being used. Each function can have its own parameter and type.

For example,

```
function int m =test(int x)
m=x;
end
function = main()
int x;
x=test(x);
end
```

GOOD

```
function =test(int x)
 x=1;
end
function = main()
test();
end
```

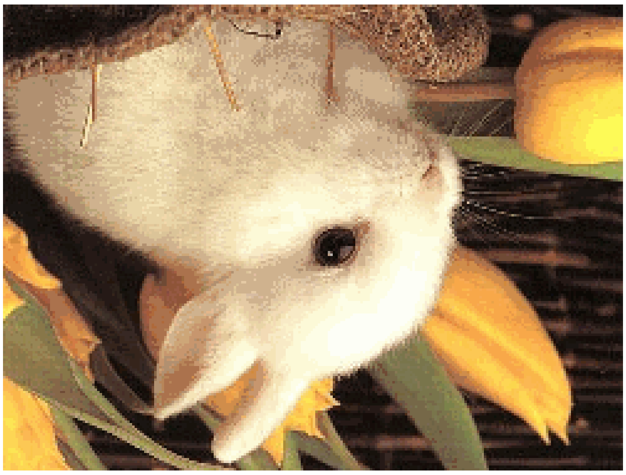
GOOD

2.6. Example

2.6.1. flip the image vertically

```
function image ret = flip(image im)
int height;
int width;
int i;
int j;
height = getheight(im);
width = getwidth(im);
ret=imnew(width,height,"RGB");
for j=0:height-1
   for i=0:width-1
      ret[i,height-j-
1,"rgb"]=im[i,j,"rgb"];
   end
end
end
```

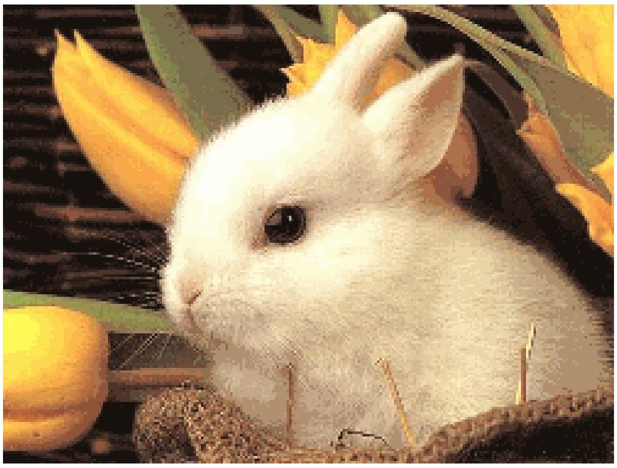
```
function = main()
  image x;
  image y;
  x=imread("./rabbit.jpg");
  imshow(x);
  y=flip(x);
  imshow(y);
 end
```



2.6.2. Flip the image horizontally

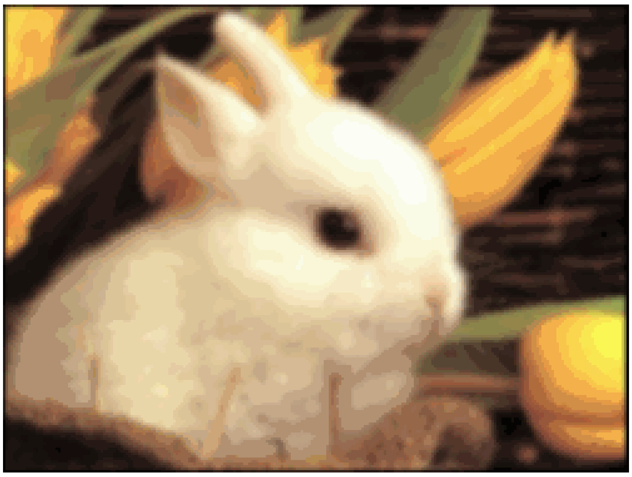
```
function image ret = flip(image im)
int height;
int width;
int i;
int j;
height = getheight(im);
width = getwidth(im);
ret=imnew(width,height,"RGB");
for j=0:height-1
    for i=0:width-1
        ret[i,height-j-
1,"rgb"]=im[i,j,"rgb"];
    end
end
end
```

```
function = main()
   image x;
   image y;
   x=imread("./rabbit.jpg");
   imshow(x);
   y=flip(x);
   imshow(y);
 end
```



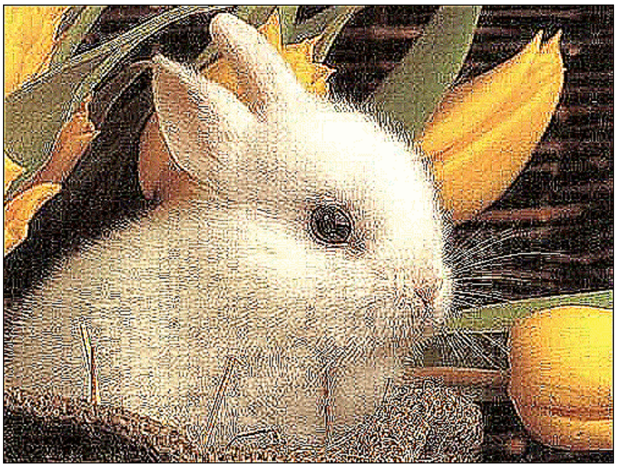
2.6.3. Blur the image

```
function = main()
   image x;
   image y;
   kernel k;
   k=
[0.25,0.0,0.25;0.0,0.0,0.0;0.25,0.0,0.2
5];
   x=imread("./rabbit.jpg");
   #x=togray(x);
   imshow(x);
   y=x@k@k@k@k@k@k@k;
   imshow(y);
   imsave(y,"./r3.gif");
end
```



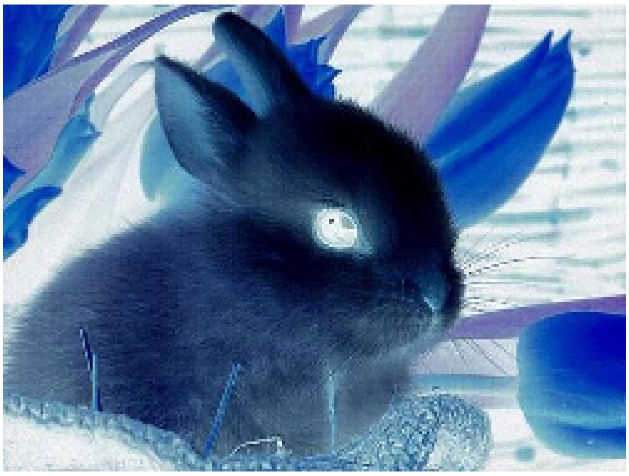
2.6.4. Sharpen the image

```
function = main()
   image x;
   image y;
   kernel k;
   k=
[0.25,0.0,0.25;0.0,0.0,0.0;0.25,0.0,0.2
5];
   x=imread("./rabbit.jpg");
   #x=togray(x);
   imshow(x);
   y=x@k@k@k@k@k@k@k;
   imshow(y);
   imsave(y,"./r3.gif");
end
```



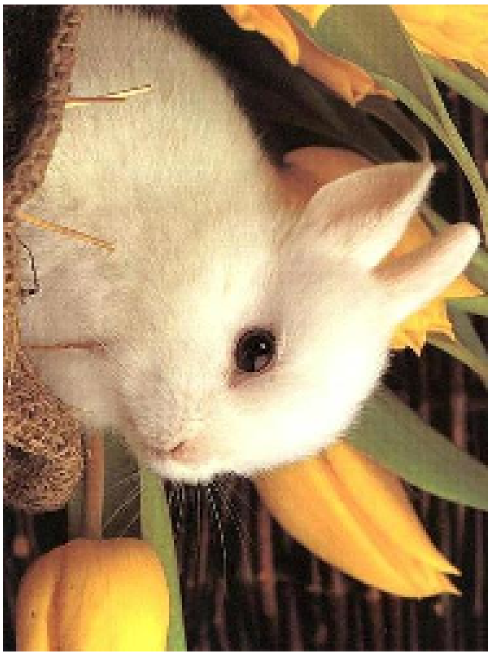
2.6.5. Inverse the image

```
function = main()
   image x;
   image y;
   kernel k;
   k=
[0.25,0.0,0.25;0.0,0.0,0.0;0.25,0.0,0.2
5];
   x=imread("./rabbit.jpg");
   #x=togray(x);
   imshow(x);
   y=x@k@k@k@k@k@k@k;
   imshow(y);
   imsave(y,"./r3.gif");
end
```



2.6.5. Rotate the image 90^0

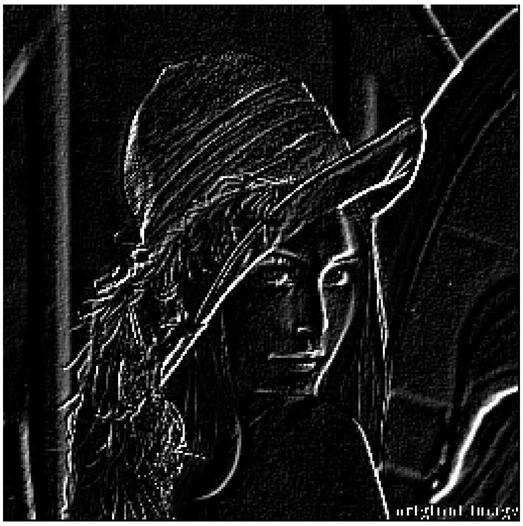
```
function image ret = rotate90(image im)
  int height;
  int width;
  int i;
  int j;
  height = getheight(im);
  width = getwidth(im);
  ret=imnew(height,width,"RGB");
  for j=0:height-1
    for i=0:width-1
            ret[height-j-1,i,"rgb"] = im[i,j,"rgb"];
    end
  end
end
function = main()
   image x;
   image y;
   x=imread("./rabbit.jpg");
   imshow(x);
   y=rotate90(x);
end
```



2.6.5. Edge Detection

```
kernel k;
int i;
int j;
function image m=edge(image b,kernel k)
  imshow(b);
  b = b@k;
  for i=0:getheight(b)
    for j=0:getwidth(b)
      if b[j,i,"grey"] < 0
        b[j,i,"grey"] = -b[i,j,"grey"];
      end
    end
  end
  imshow(b);
  imsave(b,"./lena_edge.jpg");
end
```

```
function = main()
  image a;
  image b;
  a = imread("./lena_color.jpg");
  b = togray(a);
  k = [-5.0, 0.0, 0.0;
        0.0, 0.0, 0.0;
        0.0, 0.0, 5.0];
  edge(b,k);
end
```



3. LANGUAGE MANUAL

3.1 TYPES

- **int:** The `int` data type is a 32-bit signed integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).

- **float:** The `float` data type is a single-precision 32-bit floating point. It consists of an integer part, a decimal point, a fraction part, an `e`, and an optionally signed integer exponent. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the `e` and the exponent (not both) may be missing.

- **boolean:** The `boolean` data type has only two possible values: `true` and `false`.

- **string:** The `string` data type represents a string of characters. It is internally implemented as an instance of the `String` class, but is exposed as a primitive data type in MATLIP.

- **image:** The `image` data type represents an image. Currently, only the JPEG format is supported. The `image` data type is internally implemented as an instance of the `BufferedImage` class, but is exposed as a primitive data type in MATLIP. Individual pixels are accessible using bracket annotation, `ID[x,y,"R"/"G"/"B"/"RGB"/"GRAY"]`, to access a specific color channel of the pixel represented at the coordinate (x, y).

- **kernel:** The `kernel` data type represents a kernel for image processing. It is internally implemented as an instance of the `Kernel` class, which is a matrix, but is exposed as a primitive data type in MATLIP.

3.2 LEXICAL CONVENTIONS

3.2.1 Reserved Keywords

The following is a list of reserved keywords in MATLIP. They may not be used for any other purpose.

```
if
elseif
else
end
for
while
int
float
boolean
string
image
kernel
function
and
or
not
true
false
imnew
kernelnew
```

3.2.2 Identifiers

An identifier is a sequence of alphanumeric characters and underscores. Identifiers must start with a letter, and are case sensitive.

3.3.3 Special Characters

`[]`	Brackets are used to form kernels. For example, `[0.0, -1.0, 0.0; -1.0, 5.0, -1.0; 0.0, -1.0, 0.0]` defines a kernel for image sharpening. Moreover, brackets are also used to enclose subscripts of images and kernels for accessing or assigning individual elements.
`()`	Parentheses are used to indicate precedence in arithmetic expressions. They are also used to enclose arguments of functions.
=	Used in assignment statements. Right associative.
`,`	Comma. Used to separate image and kernel subscripts, function arguments, and elements of a row in a kernel.
.	Decimal point. `314/100`, `3.14` and `.314e1` all represent the same value.
:	Colon. Used to specify `for` iterations. For-iterations can be specified by, for example, `for i=1:240.`
`;`	Semicolon. Used inside brackets to end rows for a kernel. Used after an expression to form a statement and to separate statements.
#	Pound. The pound symbol denotes a comment; it indicates a logical end of line. Any following text is ignored.
^	Power. The power symbol is used to calculate the base to the power exponent. For example, `4^3` is evaluated to `64`.
`@`	Convolution. It takes an `image` on the left and a `kernel` on the right, calculates a sum of products over an image using the kernel, and stores the results in a new image and returns it.

3.4 EXPRESSIONS

Expressions can be categorized into six types: primary expressions, unary expressions, arithmetic expressions, logical expressions, relational expressions, and assignment expressions.

3.4.1 Primary Expressions

Primary expressions can be an identifier, a literal (`true`, `false`, integer literal, float literal, string literal), an expression contained in parenthesis, an image, a kernel, an image pixel such as

```
img[i, j, "R"]
```

which evaluates to an `int`, or a kernel element such as

```
k[i, j]
```

which evaluates to a `float`.

3.4.2 Unary Expressions

A unary expression has the following form:

`unary_operator` *`expression`*

where `unary_operator` can only be the minus sign '`-`' in MATLIP, and *`expression`* can only be of type `int` or `float`.

The unary expression composed of the operator, '`-`', followed by any other primary expression such as `boolean`, `image`, `string`, or `kernel`, is illegal.

3.4.3 Arithmetic Expressions

MATLIP has two main types of arithmetic operations. The first type applies to numerical values having the type of `int` and `float`. The behavior of this type of operations is the same as that of C/C++ and Java, with an exception that both operands need to have the same type. Take `a + b` for example. If both `a` and `b` are `int`, the expression will evaluate to an integer. If both `a` and `b` are `float`, the expression will evaluate to a floating point number. If an integer and a float are mixed in an arithmetic expression, the MATLIP compiler will give an error.

The order of precedence for this type of operations, from highest to lowest, is:

- `^`
- `*` and `/`
- `+` and `-`

The second type of arithmetic operations may apply to images or kernels. For arithmetic operations (`+, -, *, /`), both operands can be either the image type or the kernel type. In addition, the second operand can be a scalar value of type `int` (for images) or type `float` (for kernels). For the convolution operation (`@`), there is a strict order where the first operand must be an image type while the second operand must be a kernel type. The following summarizes the effects of this type of arithmetic operations, in a decreasing order of precedence:

`@`	• Convolution. It takes an `image` on the left and a `kernel` on the right, calculates a sum of products over an image using the kernel, and stores the results in a new image and returns it.
`*/`	• Multiplication/Division between two images; in this case, each pixel in the first image is multiplied/divided by the corresponding pixel in the second image, and the results are stored in the new image returned by the operation. • Multiplication/Division between two kernels; in this case, each element in the first kernel is multiplied/divided by the corresponding element in the second kernel, and the results are stored in the new kernel returned by the operation. • Multiplication/Division between an image/kernel and a scalar value; in this case, individual pixels/elements are multiplied/divided by the scalar value, and the results are stored in the new image/kernel returned by the operation. For images, the scalar value must have a type of `int`. For kernels, the

	scalar value must have a type of float.
+-	• Addition/Subtraction between two images; in this case, each pixel in the second image is added/subtracted to/from the corresponding pixel in the first image, and the results are stored in the new image returned by the operation. • Addition/Subtraction between two kernels; in this case, each element in the second element is added/subtracted to/from the corresponding element in the first kernel, and the results are stored in the new kernel returned by the operation. • Addition/Subtraction between an image/kernel and a scalar value; in this case, the scalar value is added/subtracted to/from individual pixels/elements, and the results are stored in the new image/kernel returned by the operation. For images, the scalar value must have a type of int. For kernels, the scalar value must have a type of float.

3.4.4 Relational Expressions

A relational expression has the following form:

expression relational_operator *expression*

where *expression* can be of type int or float, and relational_operator can be one of the following: ‘>’, ‘<’, ‘>=’, ‘<=’, ‘==’, or ‘!=’, which represent “greater than”,” less than”, “equal or greater than”, “equal or less than”, “not equal” respectively in our language. Let A and B be two integer or float expressions. The logical expression, A>B, return true if A is greater than B else it will return false; The logical expression, A<B, return true if A is smaller than B else it will return false; The logical expression, A>=B, return true if A is equal or greater than B else it will return false; The logical expression, A<=B, return true if A is equal or smaller than B else it will return false; The logical expression, A==B, return true if A is equal to B else it will return false; The logical expression, A~=B, return true if A is not equal to B else it will return false.

The following summarizes relational operators:

>	• A>B returns true if A is greater than B else it will return false
<	• A<B returns true if A is smaller than B else it will return false
>=	• A>=B returns true if A is equal or greater than B else it will return false
<=	• A<=B returns true if A is equal or smaller than B else it will return false
==	• A==B returns true if A is equal to B else it will return false
!=	• A!=B returns true if A is not equal to B else it will return false

3.4.5 Logical Expressions

A logical expression can have the following form:

```
expression logical_operator expression
```

where *expression* can only be of type `boolean`, and `logical_operator` can be either `"and"` or `"or"`. Let `A` and `B` be two Boolean expressions. The logical expression, `A and B`, will return true if both `A` and `B` are evaluated to be `true`, or it will return `false`; The logical expression `A or B` will return `true` if at least one of `A` and `B` is evaluated to be `true`, or it will return `false`.

A logical expression can also have the following form:

```
logical_operator expression
```

where *expression* can only be of type `boolean`, and `logical_operator` can only be `"not"`. The logical expression, `not A`, will return `true` if `A` is evaluated to be `false`, or it will return `false`.

3.4.6 Assignment Expressions

An assignment expression can have the following forms:

```
identifier = expression
identifier[i, j, channel] = expression
identifier[i, j] = expression
```

The first form represents a regular assignment and evaluates to *expression*. The second form represents an image pixel assignment and evaluates to *expression*, which has a type of `int`. The third form represents a kernel element assignment and evaluates to *expression*, which has a type of `float`.

3.5 STATEMENTS

3.5.1 Expression Statements

A statement can be formed by terminating an expression with a semicolon '`;`'. Expression statements have the following form:

```
expression;
```

3.5.2 Group of Statements (Block)

A block is also a type of statement consisting of several statements. For example, a group of statements can be nested in the body of `if`-statement and a function declaration:

```
if expression
   statement+
end

function = function_name (argument_list)
```

```
   statement+
end
```

3.5.3 Variable Declaration

Variable declaration has the following form:

```
TYPE identifier;
```

In MATLIP, a variable must be declared first before it can be used. An attempt to use a variable without declaration will throw a compile error. Also, a variable name cannot be declared twice. It is also illegal to declare multiple variables in the same line. For example,

```
int x;
image A;
int x, y, z;  # This is illegal
```

Users who are used to Java/C/C++ should also note that a variable cannot be declared and assigned a value at the same line. For example,

```
int x = 3;  # This is illegal
```

When declaring a variable, MATLIP actually initializes it for the user so that a variable will never be uninitialized. The following table gives a list of variable types and the default values they are initialized to:

Data Type	Default Initial Value
`int`	`0`
`float`	`0.0`
`string`	`""`
`boolean`	`false`
`image`	100x100 black image (RGB type)

`kernel`	`[0.0, 0.0, 0.0,`
	` 0.0, 1.0, 0.0,`
	` 0.0, 0.0, 0.0]`
	(i.e. a "do-nothing" kernel)

3.5.4 Assignment Statements

An assignment statement can have the following form:

```
identifier = expression;
```

In this case, `expression` is evaluated, and then the value is assigned to `identifier`. Alternative, an assignment statement can also have the following form:

```
identifier1 = identifier2 = ... = expression;
```

In this case, since the assignment operator '=' is right associative, the right most expression is evaluated first. Therefore, the following statement:

```
ID1 = ID2 = ID3 = ID4 = expression;
```

is equivalent to

```
ID1 = (ID2 = (ID3 = (ID4 = expression)));
```

Our language can support the assignment of an identifier of a primitive type. For example:

```
int x;
int y;
string z;
image A;
x=20;
y=30;
z="R";
A=imnew(100, 200, "RGB");
A[x, y, z] = 255;
```

The element of an image is of integer type, where x and y specify the coordinate of the pixel in the image, and z specifies the base color channel to which we want to assign the value. The color channel can only be either "R", "G", "B", or "RGB" all together. In the above example, we first create a new image of type "RGB" (means color) with width equal to 100 and height equal to 200. (To create a gray-scale image, use "GREY" or "GRAY") Then, we assign the "Red" channel of the pixel having the coordinate of (20, 30) to 255.

The user can also assign a value to all the channels of a pixel at a time:

```
image A = imnew(100, 200, "RGB");

# RGB value of pixel at (20, 30) is set to 65535
# which means the Green and Blue channels are both
# assigned to 255, while the Red channel becomes 0
A[20, 30, "RGB"] = 65535;
```

Note that an image may be reassigned. In this case, the old image will be overwritten with the new image, and will be lost permanently if the user does not save the image to disk before doing the reassignment. For example, the following sample code will assign the image "lotus.jpg" to A first, and immediately overwrites it with another image "lilly.jpg".

```
image A = imread("lotus.jpg");
A = imread("lilly.jpg");
```

The assignment to a kernel type is similar to image. For example, the following code defines a kernel with a 3x3 matrix, changes the value of the element at [1, 2] to 0.1, and then overwrites the entire kernel with another 3x3 matrix:

```
kernel K ;
```

```
K = [0.0, -1.0, 0.0; -1.0, 5.0, -1.0; 0.0, -1.0, 0.0];
K[1, 2] = 0.1;
K = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9];
```

Finally, the assignment between two images/kernels will copy the image/kernel on the right to the image/kernel on the left. (Note, however, copying an image to a kernel, or vice versa, is not allowed.) Note that this is not just a reference copy operation; the image/kernel on the left will receive a new copy of the same image/kernel, and any subsequent changes to the new image/kernel will not affect the original. In the following example, A, B, and C all receives an independent copy of the image `lotus.jpg`:

```
image A;
image B;
image C;

A = B = C = imread("lotus.jpg");
```

3.5.5 Conditional

A conditional statement can have the following form:

```
if boolean_expr
   statements
end
```

In this case, if *`boolean_expr`* is evaluated to be true, then the program will execute *`statements`*. Otherwise, the program will skip *`statements`* and execute the line right after the `end` keyword.

Alternatively, a conditional statement can also have the following form:

```
if boolean_expr
   statements1
else
   statements2
end
```

In this case, if *`boolean_expr`* is evaluated to be true, then the program will execute *`statements1`*, and then jump the line after `end`. If *`boolean_expr`* is evaluated to be false, then the program will skip *`statements1`* and execute *`statements2`*.

The third form of a conditional statement is as follows:

```
if boolean_expr1
   statements1
elseif boolean_expr2
   statements2
end
```

In this case, if *boolean_expr1* is evaluated to be true, *statements1* will be executed, and then the program will jump to end. If *boolean_expr1* is evaluated to be false, then *boolean_expr2* is evaluated. If it is true, then *statements2* will be executed, and then the program will jump to end. If both *boolean_expr1* and *boolean_expr2* are evaluated to false, then the program will jump to end directly. Note that if both *boolean_expr1* and *boolean_expr2* are evaluated to be true, only *statements1* will be executed while *statements2* will be skipped.

The final form of a conditional statement is as follows:

```
if boolean_expr1
   statements1
elseif boolean_expr2
   statements2
else
   statements3
end
```

In this case, if *boolean_expr1* is evaluated to be true, *statements1* will be executed, and then the program will jump to end. If *boolean_expr1* is evaluated to be false, then *boolean_expr2* is evaluated. If it is true, then *statements2* will be executed, and then the program will jump to end. If both *boolean_expr1* and *boolean_expr2* are evaluated to false, then *statements3* will be executed. Note that if both *boolean_expr1* and *boolean_expr2* are evaluated to be true, only *statements1* will be executed while *statements2* will be skipped.

In the above examples, there is only one elseif statement. However, MATLIP supports multiple elseif statements between if and else/end, and the user may use as many elseif statements as necessary.

3.5.6 For Loops

A for-loop statement has the following form:

```
for variable = expression1 : expression2
   statements
end
```

In the above form, *expression1* is the lower bound and *expression2* is the upper bound, and *statements* in the body will be executed repeatedly while variable is incremented by one each time, until the value of variable reaches the upper bound. For example:

```
for i=1:240
   for j=1:320
      image1[i,j,"R"]=255;
   end
end
```

Alternatively, the `for`-statement can also have the following form if the user wants to specify an increment or decrement value other than one:

```
for variable = expression1 : constant : expression2
   statements
end
```

In this case, *`expression1`* is the lower bound, `constant` is the increment or decrement value, and *`expression2`* is the upper bound. **Important:** `constant` may be positive or negative, but it must be an integer or float literal and cannot be an expression containing variables or arithmetic operations. For example, the following is valid:

```
for i = 1.0 : -0.1 : 0.0
   print(i);
end
```

But the following is illegal:

```
float j;
j = -0.1;

for i = 1.0 : j : 0.0 # compiler error
   print(i);
end
```

3.5.7 While Loops

A while-loop has the following form:

```
while boolean_expr
   statements
end
```

Here, *`statements`* in the body will be executed repeatedly until the *`boolean_expr`* is evaluated to be false.

3.5.8 Function Calls

Function calls have the following form:

```
result = function_name(arguments);
```

or if there is no return value, simply

```
function_name(arguments);
```

where *result* is an identifier bound to a variable that is used to store the return value, *function_name* is an identifier bound to the function name, and *arguments* is a list of zero or more expressions separated by commas (,).

3.6 FUNCTION DEFINITIONS

Function definitions have the following form:

```
function ret_type ret_value = function_name(argument_list)
     statements
end
```

where **function** and **end** are keywords, *function_name* is an identifier, *argument_list* is a list of zero or more arguments having the form of (TYPE ID) separated by commas (,), *ret_type* is the type of the return value, and *ret_value* is a local variable where the return value is stored. If the function does not have a return value, *ret_type* and *ret_value* should be omitted.

For example, the following function calculates the average of two floats:

```
function float answer = avg(float x, float y)
    float sum = x + y;
    answer = sum/2;
end
```

Here, the function name is avg; answer is the return value; and x and y are the arguments taken by the function.

A note on recursive functions: MATLIP supports recursion. That is, a function may call itself in the body. However, mutual recursion is not supported by MATLIP.

3.7 BUILT-IN FUNCTIONS

MATLIP provides some useful built-in functions, as described below:

3.7.1 imread(string path)

imread() is used to load an image from the disk to the program. It takes as argument a string, which specifies the pathname of the image on disk, and returns an image object. Currently, only the JPEG format is supported. For example, the following code fetches an image located at /MATLIP/images/lotus.jpg and loads it to the variable A:

```
image A;
A = imread("/MATLIP/images/lotus.jpg");
```

3.7.2 imsave(image im, string path)

`imsave()` is used to save a given image to the disk. It takes as arguments an `image` to be saved, and a `string` which specifies the path and file name of the file to be saved. The function does not return a value. Currently, only the JPEG format is supported. For example, the following code fetches an image located at `/MATLIP/images/lotus.jpg`, applies a kernel to it, and saves the new image at `/MATLIP/images/lotus_2.jpg`:

```
image A;
image B;
kernel K;

A = imread("/MATLIP/images/lotus.jpg");
K = [0.0, -1.0, 0.0; -1.0, 5.0, -1.0; 0.0, -1.0, 0.0];
B = A @ K; # apply kernel to image
imsave(B, "/MATLIP/images/lotus_2.jpg");
```

3.7.3 imshow(image im)

`imshow()` is used to display an image on the screen. It takes as arguments an `image` to be displayed, and pops up a window to show the image. The function does not return a value; it is internally implemented using the `JFrame` class. The function is non-blocking; it can be invoked for multiple times and multiple windows will pop up to display images. For example, the following code fetches an image from disk, displays it on the screen, applies a kernel to it, and displays the resulting image on the screen:

```
image A;
image B;
kernel K;

A = imread("/MATLIP/images/lotus.jpg");
imshow(A);
K = [0.0, -1.0, 0.0; -1.0, 5.0, -1.0; 0.0, -1.0, 0.0];
B = A @ K; # apply kernel to image
imshow(B);
```

3.7.4 imnew(int width, int height, string type)

`imnew()` is used to create a new image with a specified width and height. It takes as arguments an `integer` for width, an `integer` for height, and a `string` that specifies the type of the image, and returns the new image. The type can be either `"RGB"`, meaning it is a color image, or `"GREY"` or `"GRAY"`, meaning it is a grey-scale image. For example, the following code creates a color image with width 100 and height 200:

```
image A;
A = imnew(100, 200, "RGB");
```

3.7.5 print(data)

`print()` is used to print data onto the console. It takes as arguments one or more expressions of any primitive data types except `image` and k`ernel`, and prints them onto the console. The function does not return a value. Individual expressions can be concatenated using the plus '`+`' operator. For example, the following code prints onto the consol "`Hello John, you are 27 years old!`":

```
string name;
int age;
name = "John";
age = 27;
print("Hello " + name + ", you are " + age + " years old!");
```

3.7.6 kernelnew(int width, int height)

`kernelnew()` is used to create a kernel without immediately specifying the value of each element. It is especially useful when the size of kernel is large. For example, creating a kernel with both width and height equal to 9 would require user to specify the values of 81 elements. `kernelnew()` will initialize each element of the newly created kernel to `0.0` and allows the user to specify the values later. The following example shows a typical usage of `kernelnew()`:

```
kernel k;
int i;
int j;
k=kernelnew(9,9);
for j=0:8
   for i=0:8
      k[i,j]=1.0/9.0; # a blurring kernel
end
```

3.7.7 read()

`read()` takes the user input from the consol and returns a `string`. For example,

```
image A;
string path;
path=read();
A=imread(path);
```

3.7.8 toint(data)

`toint()` converts the parameter to the type of integer. The argument "data" can be the type of either `string` or `float`. For example,

```
int height;
int width;
image A;
width=toint(read());  # read width from user
height=toint(read()); # read height from user
x=imnew(width,height,"RGB");
imshow(x);
```

3.7.9 tofloat(data)

`tofloat()` converts the parameter to the type of float. The argument “data” can be the type of either `string` or `int`. For example,

```
int a;
float value;
a=100;
value=tofloat(a);
```

3.7.10 tostring(data)

`tostring()` converts the parameter to the type of string. The argument “data” can be the type of either `integer` or `float`. For example,

```
int age;
string s;
age=30;
s = "Joe is " + tostring(age) + " years old."
print(s);
```

3.7.11 togray(image im)

`togray()` is used to convert a color image to a gray-scale one. If the input image is already a gray-scale one, the function has no effect. For example,

```
image im;
image im2;
im=imread("./rabbit.jpg");
im2=togray(im);
imshow(im2);
```

3.7.12 sqrt(data)

sqrt() is used to calculate the square root of a value. The argument “data” can be the type of either `int` or `float`. The return value will have the same type as the argument. For example,

```
float a;
a = 43.56;
print(sqrt(a)); #will print 6.6
```

3.7.13 mod(a, n)

`mod(a, n)` is used to calculate the modulo of two numbers; that is, the remainder on division of `a` by `n`. The arguments can be tye type of either `int` or `float`, but `a` and `n` must have the same type. The return value will have the same type as the arguments. For example,

```
float a;
float b;
```

```
a = 7.7;
b = 3.3;
print(mod(a,b)); #will print 1.1
```

3.7.13 Get Functions

MATLIP also supports some built-in “get” functions to help the user with image manipulation. The following briefly describes each of these functions:

- **getwidth(image im | kernel k)**: takes as arguments an image or a kernel, and returns the width of the image/kernel as an integer.
- **getheight(image im | kernel k)**: takes as arguments an image or a kernel, and returns the height of the image/kernel as an integer.
- **gettype(image im)**: takes as argument an image, and returns the type of the image as a string. The type can be either "RGB" or "GREY", which indicates a color image or a grey-scale image, respectively.
- **getlength(string s)**: takes as argument a string, and returns the length of the string as an integer.

3.8 SCOPE & NAMES

3.8.1 Static Scoping

MATLIP uses static scoping. That is, the scope of a variable is a function of the program text and is unrelated to the runtime call stack. In MATLIP, the scope of a variable is the most immediately enclosing block, excluding any enclosed blocks where the variable has been re-declared.

3.8.2 Global vs. Local

Global variable: The variables declared outside of the function are global variables, which will be applied in the whole program except the function where there is a local variable with the same name as that of the global variable. Global variable will exist until the program terminates.

Local variable: The variables declared inside of the function are local variables, which will exist and be applied only inside that function.

Scope conflicts: If there is a global variable whose name is the same with that of the local variable, then the value of the local variable will be applied inside the function while the value of the global variable will be applied in all the other part of the program except that function.

For example:

```
int i;
```

```
i = 0;  # global variable i with the value 0

function = f(int i)
   i = i + 1; # local variable with value 3 when f(2) is called
end #end of scope of local variable i

f(2);
i = i + 1; # global variable i unchanged, now with a value 1
```

3.8.3 Forward Declarations

MATLIP requires forward declarations for variables. That is, a variable needs to be declared before it can be referenced. In addition, all variable declaration statements must precede other types of statements in a function. A variable declaration statement in the middle of a function body is not valid.

Function declarations, however, do not have this constraint. Functions in MATLIP are free-form, meaning that they can be defined anywhere in the program text. A function may be invoked before its declaration. Function declarations inside another function body, however, is not allowed.

The following code fragment is perfectly valid and shows that MATLIP allows a function can be invoked before its declaration:

```
function = main()
   float a;
   float b;
   float mean;

   mean = func(a, b);
end

function float answer = func(float x, float y)
    ...
end
```

3.8.4 Arithmetic Operator Overloading

Arithmetic operators (`+, -, *, /`) are overloaded in MATLIP. They can be used in expressions with integers and floats. They can also be used between images or kernels, and between images/kernels and scalar values. These operations are described in Section 3.4.3: Arithmetic Expressions.

The convolution operator (`@`), however, is not overloaded. It takes exactly an `image` on the left and a `kernel` on the right, nothing else.

3.8.5 Function Name Overloading

MATLIP does not allow function name overloading. That is, each function should have a unique function name, or MATLIP compiler will complain. This helps make a MATLIP program more readable and easier to understand, which are two important goals of the language.

3.8.6 Namespaces

MATLIP has multiple namespaces, one for global functions, one for global variables, and one for local variables in each function. That is, functions and variables, and global variables and local variables, can all share the same names and MATLIP is clever enough to distinguish them. This is, however, not recommended as it only makes a program less readable and harder to understand.

4. PROJECT PLAN

4.1 Process

The process we used for project planning was we met on every Friday, proposed uncertainties, and tried to come up with solutions. If we could not come up with any possible solutions, we would either drop by at TA's hour or send an email to TA to seek solutions. In addition, we spent at least two days in a week working on the project together in clic lab. Since we tried to divide the job as less dependent as possible, we were able to work simultaneously. This process basically executes and continues until the end of the project.

4.2 Programming Style

4.2.1 Java

We try to make each of our Java method names as intuitive as possible. Method names in Java usually start with a lower case verb, followed by one or more nouns with the first letters capitalized. For example: `doArithmetic()`, `setImagePixel()`, `getType()`, etc. Some of the Java methods share the same name as MATLIP built-in functions, such as `imread()` and `kernelnew()`. This makes function mapping more straightforward in `javaprinter.ml`.

Also, we chose to align the starting curly brace at the end of the declaration of a method, and the end brace on the left side of the last line of a block. For example:

```
static float getKernelData(...){
    ....
}
```

4.2.2 OCaml

Since OCaml is new to us, we chose to follow the programming styles and conventions presented in the MicroC example. For instance, the OR operators ‘`|`’ are aligned vertically, and all the lines belonging to the same block are also aligned vertically to the left. When hierarchy exists, we use two spaces to distinguish between each level of hierarchy. We avoid using tabs because different editors have different spacing for tabs.

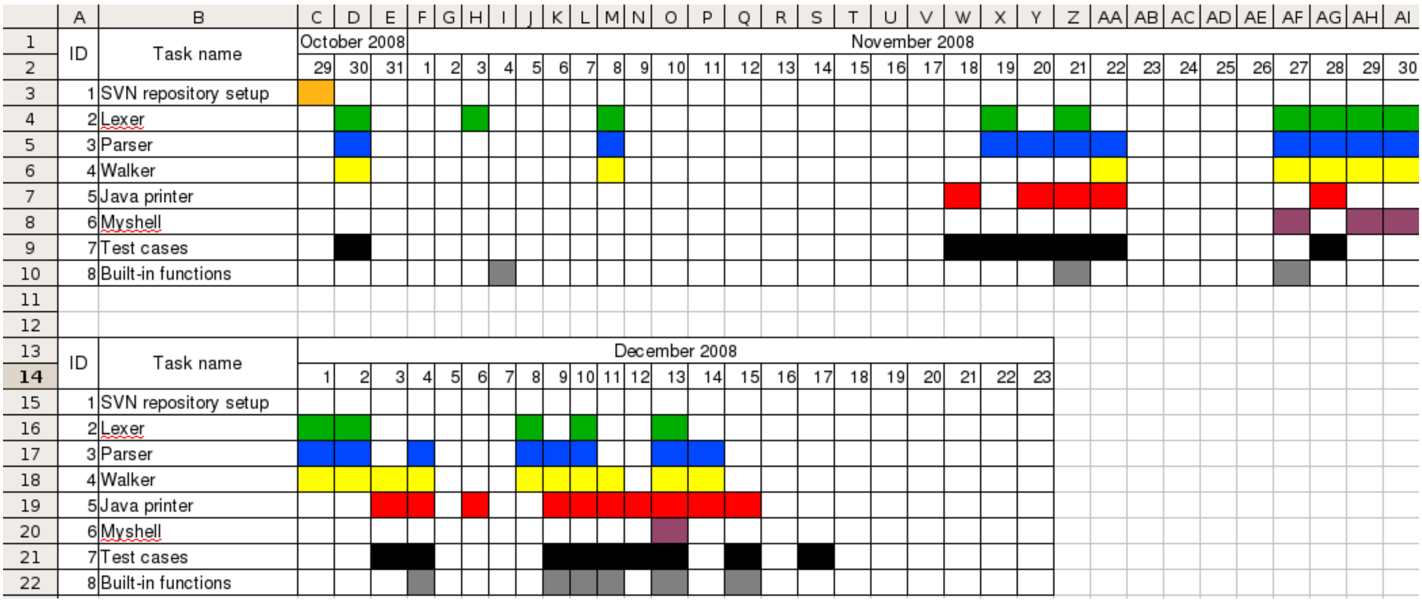
For function declarations, we also try to make the names as intuitive as possible. In OCaml, we use underscores to separate words in a function name. The naming convention is usually *`verb_noun`* or *`noun_of_noun`*. We also try to give a descriptive comment before each function declaration, briefly describing what the function does and what arguments it takes.

The following code fragments illustrate these points:

```
(* given variable v, find it in symbol table *)
let rec find_symbol symbols v =
  (* try to find it in local table first *)
  let exists = List.exists (fun a -> if a.varname = v then true else false) symbols in
  if exists then
    let var = List.find (fun a -> if a.varname = v then true else false) symbols in
    var
  else (* then try to find it in global table *)
    let exists2 = List.exists (fun a -> if a.varname = v ^ "_global_var" then true
    else false) symbols in
    if exists2 then
      let var = List.find (fun a -> if a.varname = v ^ "_global_var" then true else
       false) symbols in
      { varname = v; vartype = var.vartype }
    else
      raise (Failure ("Unable to find " ^ v ^ " in symbol table"))

(* returns the type of an expression, mainly for images, kernels, and power operator
*)
let rec type_of_expr symbols = function
    Literal(l) -> { varname = string_of_int l; vartype = "int" }
  | Floatlit(f) -> { varname = string_of_float f; vartype = "float" }
  | String(s) -> { varname = s; vartype = "string" }
  | Bool(s) -> { varname = s; vartype = "boolean" }
  | Id(s) ->
      (* try to find it in local table first *)
      let exists = List.exists (fun a -> if a.varname = s then true else false)
       symbols in
      if exists then
        let var = List.find (fun a -> if a.varname = s then true else false) symbols
         in
        var
      else (* then try to find it in global table *)
        let exists2 = List.exists (fun a -> if a.varname = s ^ "_global_var" then true
         else false) symbols in
        if exists2 then
          let var = List.find (fun a -> if a.varname = v ^ "_global_var" then true
          else false) symbols in
          { varname = s; vartype = var.vartype }
        else
          raise (Failure ("Unable to find " ^ s ^ " in symbol table"))
```

4.3 Timeline

Color block indicates the accomplishment of specific tasks. The timeline chart is built based on the committed log of subversion.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI
1	ID	Task name	October 2008			November 2008																													
2			29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
3	1	SVN repository setup	🟧																																
4	2	Lexer		🟩				🟩					🟩											🟩		🟩						🟩	🟩	🟩	🟩
5	3	Parser		🟦									🟦											🟦	🟦	🟦	🟦					🟦	🟦	🟦	🟦
6	4	Walker		🟨									🟨														🟨					🟨	🟨	🟨	🟨
7	5	Java printer																					🟥		🟥	🟥	🟥						🟥		
8	6	Myshell																														🟪		🟪	🟪
9	7	Test cases		⬛																			⬛	⬛	⬛	⬛	⬛						⬛		
10	8	Built-in functions							▩																	▩						▩			
11																																			
12																																			
13	ID	Task name	December 2008																																
14			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23										
15	1	SVN repository setup																																	
16	2	Lexer	🟩	🟩						🟩		🟩			🟩																				
17	3	Parser	🟦	🟦		🟦				🟦	🟦	🟦			🟦	🟦																			
18	4	Walker	🟨	🟨	🟨	🟨				🟨	🟨	🟨	🟨		🟨	🟨																			
19	5	Java printer			🟥	🟥		🟥			🟥	🟥	🟥	🟥	🟥	🟥	🟥																		
20	6	Myshell													🟪																				
21	7	Test cases			⬛	⬛					⬛	⬛	⬛	⬛	⬛		⬛		⬛																
22	8	Built-in functions				▩					▩	▩	▩		▩		▩																		

4.4 Roles and Responsibilities

Member	Roles & Responsibilities
Pin-Chin Huang	**Back-End Lead:** Lexer/Parser, AST, part of Walker, JavaPrinter, Documentation
Shariar Zaber Kazi	**Front-End Lead:** Lexer/Parser, AST, most of Walker, Documentation
Shih-Hao Liao	**Testing Lead:** Test Files, Matlip Testing, Shell Scripts, Presentation Slides, Documentation
PoHsu Yeh	**Java Lead:** Java Library design & implementation, Image Processing Algorithms, Shell Scripts, Demo code, Documentation

4.5 Tools and Languages

We use gedit as the development tool. Ocaml is the language we use to write lexer, parser, walker, and Java printer. Further, we use shell script to build the test suite.

4.6 Project log

```
12/18/08:
23:09 Changeset [192] by pinchin1981
updated setImagePixel() again
23:07 Changeset [191] by hendry
added edge detection case
23:05 Changeset [190] by pinchin1981
updated setImagePixel()
23:00 Changeset [189] by pinchin1981
added exit() to getImagePixel() and setImagePixel()
22:57 Changeset [188] by hendry
added lena_edge.jpg
22:56 Changeset [187] by hendry
added lena.jpg
22:48 Changeset [186] by hendry
added golf_edge
22:44 Changeset [185] by pinchin1981
added image detection code, added golf.jpg
13:01 Changeset [184] by pinchin1981
fixed namespace bug in javaprinter, updated test file
12:30 Changeset [183] by pinchin1981
modified javaprinter to distinguish between global functions, global variables, and local variables
11:15 Changeset [182] by pinchin1981
added check for duplicate names in javaprinter: functions and variables cannot share the same name. added test file
10:50 Changeset [181] by pinchin1981
changed image type of imnew() to 3BYTE_BGR
10:46 Changeset [180] by hendry
updated imnew method
10:31 Changeset [179] by pinchin1981
updated doArithmetic() to solve JPEG image saving problem
10:28 Changeset [178] by hendry
updated doArithmetic method
06:07 Changeset [177] by bottle
added test case of image inverse
05:31 Changeset [176] by pinchin1981
updated setImagePixel() and getImagePixel() to fix the alpha channel problem
05:28 Changeset [175] by pinchin1981
changed javaprinter's Convolve() to use a java method instead
05:05 Changeset [174] by pinchin1981
incorporated hendry's java fixes for image processing into javaprinter.ml
04:57 Changeset [173] by hendry
updated doArithmetic method
03:51 Changeset [172] by hendry
updated doArithmetic method
```

03:43 Changeset [171] by pinchin1981
CONVOLVE should have a higher precedence than TIMES/DIVIDE per our LRM
03:33 Changeset [170] by hendry
added convolve method
03:25 Changeset [169] by pinchin1981
fixed consecutive image/kernel cloning problem.. now im1=im2=im3=imread() and k1=k2=k3=kernelnew() are supported. added test file
03:22 Changeset [168] by pinchin1981
change '=' token to right associative
02:46 Changeset [167] by pinchin1981
let walker.ml allow kernel op float
02:41 Changeset [166] by pinchin1981
let walker.ml block int op image
12/17/08:
07:55 Changeset [165] by hendry
added test case of read()
07:20 Changeset [164] by hendry
added a gif image
03:34 Changeset [163] by hendry
added test cases of image blur and sharpen
12/15/08:
02:54 Changeset [162] by hendry
changed the absolute path to relative path of all image test cases
02:37 Changeset [161] by pinchin1981
updated setImagePixel() method
02:11 Changeset [160] by hendry
updated getImagePixel, setImagePixel
02:00 Changeset [159] by pinchin1981
added Hendry's getImagePixel() and setImagePixel() java methods to javaprinter, updated java imnew() method
01:39 Changeset [158] by hendry
added setImagePixel and getImagePixel methods and updated imnew method
12/14/08:
19:35 Changeset [157] by pinchin1981
updated AssignPixel and ImageAccess in javaprinter according to new grammar, slightly modified Makefile
01:08 Changeset [156] by shariar
Changes pixel assignment, and pixel access grammer as per Patrick.
12/13/08:
06:41 Changeset [155] by shariar
Added function togray() which takes an image and returns an image.
06:05 Changeset [154] by pinchin1981
added togray() function that transforms a color image to a gray-scale one, updated test file
05:43 Changeset [153] by pinchin1981
implemented sqrt() in javaprinter, added test file
05:28 Changeset [152] by pinchin1981
implemented read() in javaprinter, added test file
05:22 Changeset [151] by pinchin1981
implemented toint(), tofloat(), tostring() in javaprinter.ml; added test file
05:18 Changeset [150] by hendry
added read method
04:50 Changeset [149] by pinchin1981
implemented mod() in javaprinter, added test file
04:49 Changeset [148] by shariar
Added function redefined checker so that a function with the same name cannot occur twice.
04:26 Changeset [147] by pinchin1981
added fuller expr type checker in javaprinter, implemented power operator, added test file
04:05 Changeset [146] by hendry
updated objectToFloat and objectToInt
04:05 Changeset [145] by shariar
Added built-in function sqrt() which takes either an int or float and returns the corresponding square root of
03:55 Changeset [144] by shariar
Added tofloat() function which is similar to toint().
03:51 Changeset [143] by shariar
Added built-in function toint() which takes either a float/string and returns an integer.
03:45 Changeset [142] by shariar
Added read() built-in function which does not take any parameter and returns a string as user input.
03:40 Changeset [141] by hendry
update makefile

```
03:38 Changeset [140] by shariar
Added mod() function which operates on either float, or int and the corresponding return type.
03:31 Changeset [139] by hendry
update makefile
03:21 Changeset [138] by hendry
delete testall.sh interpret.ml
03:17 Changeset [137] by hendry
update myshell.sh
03:12 Changeset [136] by hendry
update makefile, ast.mli
03:07 Changeset [135] by hendry
rename microc.ml to matlip.ml
03:06 Changeset [134] by hendry
rename microc.ml to matlip.ml
03:04 Changeset [133] by hendry
rename printer.ml to walker.ml
02:46 Changeset [132] by hendry
added test cases ofhorizontal and vertical flip
02:29 Changeset [131] by shariar
Fixed floating number rule to the right one. Added power operator "".
00:10 Changeset [130] by hendry
fixed the bug of image rotation
12/12/08:
23:40 Changeset [129] by hendry
added a test case for rotating an image to 90 degree
05:07 Changeset [128] by pinchin1981
added support for grey scale images. For accessing or assigning pixels for a gray-scale image x, use x[i,j,"grey"] or x[i,j,"gray"]
12/11/08:
22:54 Changeset [127] by hendry
added java code to convert color image to grey scale
22:49 Changeset [126] by pinchin1981
updated Java clone() method, modified printJava.ml, updated test case. Now when doing im2 = im1; and k2 = k1; im2 and k2 should receive their own copy of object, not just a reference
22:31 Changeset [125] by pinchin1981
updated java getType() method
20:46 Changeset [124] by hendry
fixed the bug on getType() method
08:13 Changeset [123] by pinchin1981
added gettype() to printer.ml and printJava.ml, updated test file
07:55 Changeset [122] by pinchin1981
added getwidth(), getheight(), getlength() for images/kernels/strings in printJava, added test file
07:36 Changeset [121] by pinchin1981
added boolean support in printJava; added test file
07:14 Changeset [120] by pinchin1981
added AssignKernel (for assigning individual kernel elements), modified test case
06:06 Changeset [119] by pinchin1981
updated printJava for expr @ expr, added And, Or, Not operators, unable to test images from home but generated Java code compiles OK, Joe please test
04:28 Changeset [118] by pinchin1981
dd
04:27 Changeset [117] by pinchin1981
deleted more
04:26 Changeset [116] by pinchin1981
deleted unnecessary backup files
12/10/08:
20:30 Changeset [115] by hendry
added support for kernel cloning
14:37 Changeset [114] by shariar
Fixed the bugs that Joe reported. Added and, or, not operators. Fixed a few more bugs.
08:23 Changeset [113] by bottle
add new test cases
06:46 Changeset [112] by pinchin1981
let printJava ignore statements that don't have an lvalue to store the result of an expression
05:02 Changeset [111] by pinchin1981
fixed major bug in printJava
04:13 Changeset [110] by pinchin1981
```

added AssignPixel?() back in printJava
01:19 Changeset [109] by pinchin1981
raise exception when attempting image/kernel arithmetic operations other than +, -, *, /
01:18 Changeset [108] by bottle
add test case
12/09/08:
22:34 Changeset [107] by shariar
Added kernel assignment similar to pixel assignment. example: kernel …
22:22 Changeset [106] by shariar
Added back again pixel access and fixed a little bug.
07:35 Changeset [105] by pinchin1981
added kernelnew(), added kernel access, added image convolution, added kernel init, fixed function bugs (formals and return variables were not in symbol table), added test files, added gif image
06:04 Changeset [104] by bottle
add test case
05:24 Changeset [103] by pinchin1981
fixed minor grammar error
05:04 Changeset [102] by hendry
added kernelinit()
04:50 Changeset [101] by hendry
added getKernelData and setKernelData
03:46 Changeset [100] by hendry
modify kernelnew method
03:12 Changeset [99] by pinchin1981
forgot to add test file
03:09 Changeset [98] by pinchin1981
in printJava.ml: added global functions to symbol table, added expression type checking, fixed image/kernel arithmetic operation bugs, added kitten.jpg
02:09 Changeset [97] by shariar
added boolean support for if/else/while loop.
12/08/08:
23:21 Changeset [96] by shariar
Gray image support in accessing image pixel
23:01 Changeset [95] by shariar
Fixed grammer and added boolean type
12/06/08:
05:14 Changeset [94] by pinchin1981
fixed bin op match failure bug
12/04/08:
10:27 Changeset [93] by pinchin1981
add support for image cloning (image assignment)
10:13 Changeset [92] by hendry
added image clone method
10:07 Changeset [91] by pinchin1981
add support for consecutive image/kernel arithmetic operations; now we can do a = x + y + z + w; all of them being images
09:43 Changeset [90] by pinchin1981
add image/kernel arithmetic operations, add image cloning mechanism, add test file
08:42 Changeset [89] by hendry
clone image
08:06 Changeset [88] by hendry
changed imnew. this version is correct
08:03 Changeset [87] by hendry
changed imnew
07:46 Changeset [86] by hendry
added clone image syntax
07:23 Changeset [85] by pinchin1981
add symbol table mechanism to printJava.…
07:01 Changeset [84] by hendry
added doArithmetic method
04:12 Changeset [83] by pinchin1981
add support for assigning pixel values, add test file
03:16 Changeset [82] by hendry
added doArithmetic
03:12 Changeset [81] by pinchin1981
add image access support in printJava.ml, add test file
03:08 Changeset [80] by bottle

```
add pixel assignment
02:59 Changeset [79] by bottle
add A[10,10,rgb]=5
02:22 Changeset [78] by pinchin1981
ImageAccess?() channel should be r, g, b, or rgb
12/03/08:
23:22 Changeset [77] by pinchin1981
add check: width and height in imnew() cannot be zero or negative (we can check these only if they're literals, though)
22:38 Changeset [76] by bottle
add test cases
22:32 Changeset [75] by pinchin1981
add image support to printJava.ml, modify printer.ml to allow …
22:01 Changeset [74] by bottle
add test case
12/02/08:
13:47 Changeset [73] by shariar
Added convolve (@) operation support.
12:32 Changeset [72] by shariar
Moved imnew(), and kernelnew() to the right place.
12:24 Changeset [71] by shariar
Added kernel type initialization like array. if I have the following …
12:13 Changeset [70] by shariar
string literals cannot have any new line inside them (java syntax error).
12/01/08:
10:11 Changeset [69] by shariar
Added kernel type, kernelnew(), and kernel access. Still to add: kernel initialization.
05:08 Changeset [68] by hendry
added java code for +-*/ operations to images
11/30/08:
12:09 Changeset [67] by shariar
Added improved error reporting in scanner.
11:48 Changeset [66] by shariar
Added getheight, getwidth, and getlength functions.
11:26 Changeset [65] by shariar
Added fully-functional print method. Added built-in function imshow().
07:56 Changeset [64] by shariar
Added built-in function imsave(im, "image-path")
07:18 Changeset [63] by shariar
Added built-in function imread("image-path").
04:21 Changeset [62] by shariar
Added newImage() support. I assumed the following syntax:
02:22 Changeset [61] by bottle
add error=0 after each loop
11/29/08:
22:38 Changeset [60] by hendry
rename testing
22:16 Changeset [59] by hendry
update shell
21:54 Changeset [58] by hendry
rename testing
16:20 Changeset [57] by shariar
Added full support for image access grammar. Added full support for image access
Lexer now keeps track of line count for each token.
Parser now prints line number of any syntax, and continues parsing the file.
11/28/08:
12:30 Changeset [56] by pinchin1981
add two test files for 'elseif'
12:29 Changeset [55] by pinchin1981
add 'elseif' to scanner, parser, printer.ml, and printJava.ml... wasn't easy!
08:24 Changeset [54] by pinchin1981
fixed string bugs for functions and global variables
07:15 Changeset [53] by pinchin1981
add code to prevent users from applying operators other than '+' '==' '!=' to string
06:05 Changeset [52] by pinchin1981
analyzer (printer.ml) now allows for string concatenation with other data types except image and kernel
04:49 Changeset [51] by pinchin1981
```

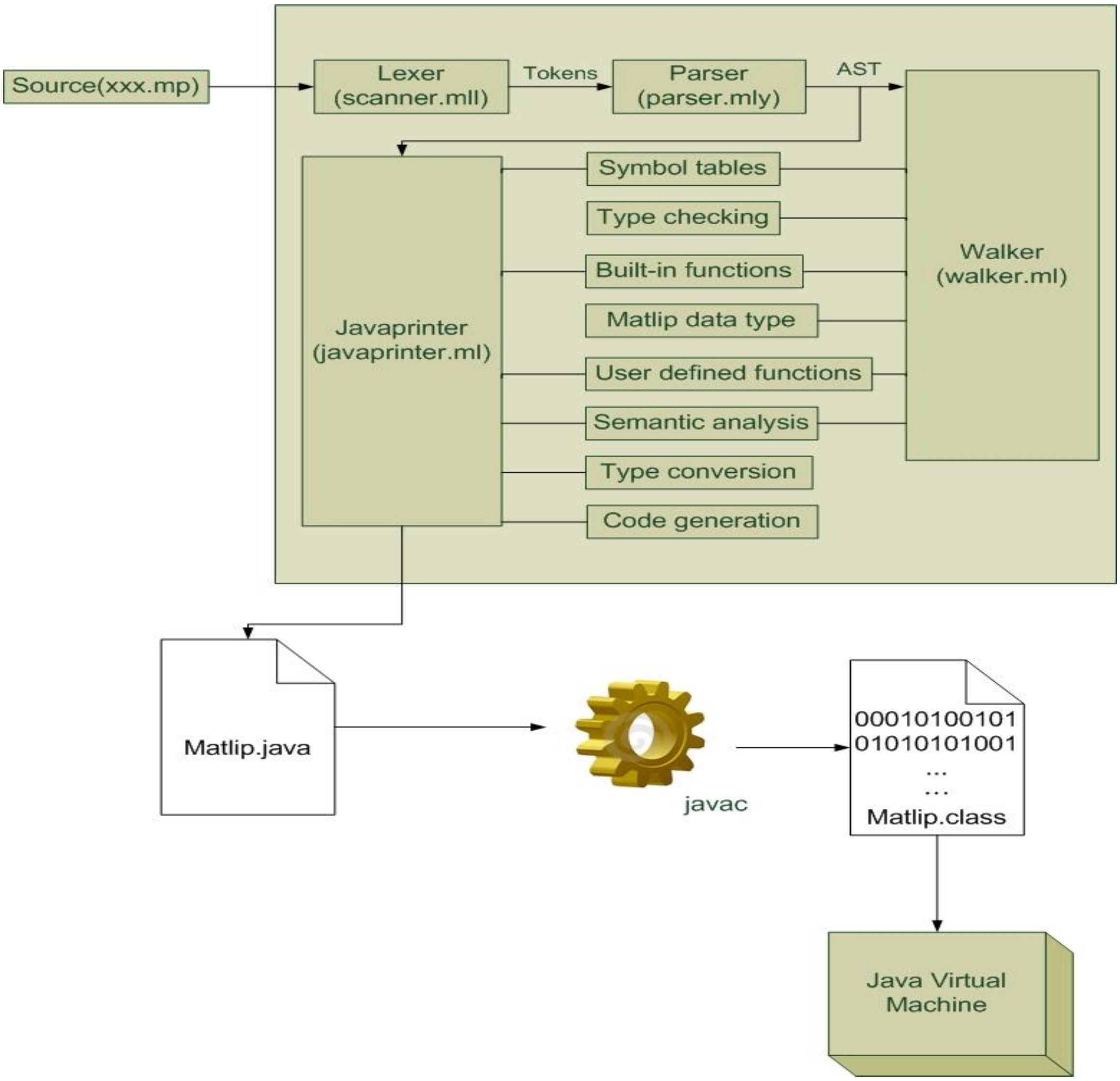
```
added string support in printJava.ml, added type conversion
11/27/08:
23:52 Changeset [50] by shariar
Draft image type access support
22:34 Changeset [49] by shariar
--
21:24 Changeset [48] by hendry
upload myshell
21:14 Changeset [47] by hendry
addd java code for image processing and corresponding syntax comments
21:09 Changeset [46] by shariar
Added string definition and handling.
21:08 Changeset [45] by shariar
Added string type in lexer
03:54 Changeset [44] by hendry
java code which can do


    1. apply kernel to an image for sharpen and blur
    2. set/get pixel value to/from an image
    3. set/get value to/from an image channel
    4. set/get value to/from kernel 11/22/08:
13:09 Changeset [43] by shariar
Bug fixed & for/while/if statement support added.
12:00 Changeset [42] by pinchin1981
we don't need temp in microc
11:59 Changeset [41] by bottle
add for test case
11:45 Changeset [40] by hendry
done the shell script which is myshell.sh
11:32 Changeset [39] by bottle
add for test case
11:03 Changeset [38] by shariar
Makefile for printer; excludes printerJava
10:42 Changeset [37] by shariar
Added nested recursive function calls from within a parameter.
10:18 Changeset [36] by bottle
modify test file
10:14 Changeset [35] by bottle
add test file
09:43 Changeset [34] by bottle
add test file
09:12 Changeset [33] by pinchin1981
changed function name…
09:08 Changeset [32] by pinchin1981
added variable initialization for local and global variables
09:02 Changeset [31] by bottle
add test/fail file
08:33 Changeset [30] by pinchin1981
added negative increment value for our for-loop, modified some test files
08:14 Changeset [29] by shariar
--
08:12 Changeset [28] by shariar
Removed all parenthesis.
08:10 Changeset [27] by shariar
Removed parenthesis from for loop.
08:08 Changeset [26] by bottle
added some test files
07:57 Changeset [25] by shariar
Added datatype.
07:57 Changeset [24] by shariar
Added datatype support.
07:56 Changeset [23] by shariar
Added full function support and error checking.
07:06 Changeset [22] by pinchin1981
added unary minus, added test files
05:59 Changeset [21] by pinchin1981
```

changed the way RETURN works, fixed non-static variable error, added a test file
04:01 Changeset [20] by pinchin1981
added hendry's image code to printJava.ml, added and modified some test files
11/21/08:
22:59 Changeset [19] by hendry
changed the built-in functions to static
06:57 Changeset [18] by pinchin1981
fixed S/R conflicts, added function declaration to printJava.ml
05:00 Changeset [17] by pinchin1981
resolved one of the S/R conflicts.. two remaining
02:24 Changeset [16] by shariar
Added a few keywords.
02:23 Changeset [15] by shariar
Added function and datatype declaration.
02:22 Changeset [14] by shariar
Added, and modified data types
11/20/08:
23:58 Changeset [13] by pinchin1981
remove parenthesis for IF and WHILE statements, add test file
23:42 Changeset [12] by pinchin1981
add new for loop in printJava.ml, add new test files
11/19/08:
05:25 Changeset [11] by pinchin1981
delete unnecessary files from repository
05:14 Changeset [10] by pinchin1981
add END token, remove braces for IF, FOR, WHILE statements, updated some test files
11/18/08:
22:45 Changeset [9] by pinchin1981
added printJava.ml to generate Java code, modified microc.ml to invoke printJava as 2nd pass
11/08/08:
10:14 Changeset [8] by shariar
Implemented for loop. Cleaned/removed some code.
06:56 Changeset [7] by shariar
Changed printer.ml Adding for loop syntax of our language
11/04/08:
21:30 Changeset [6] by hendry
done the built-in functions of newImage, print, and get functions
00:46 Changeset [5] by hendry
done the built-in functions: imread, imshow, and imsave. I will start to …
00:46 Changeset [4] by hendry
done the built-in functions: imread, imshow, and imsave. I will start to …
11/03/08:
07:20 Changeset [3] by hendry
added regular expressions for keywords and special characters in lexer
10/30/08:
21:42 Changeset [2] by bottle
change scanner
21:29 Changeset [1] by hendry
the initial version

5. ARCHITECTUAL DESIGN

We implemented lexer, parser, walker, and java printer by OCaml. The lexer(scanner.mll) takes the input program and outputs a stream of tokens to parser. Parser(parser.mly) parses those tokens and builds an unambiguous abstract syntax tree. Then the walker walks through the tree, builds symbol table and checks type. Then Java printer walks through the tree again and prints out the java code.

5.1 Block Diagram



5.2 Data Types

Maplip has seven primitive data types: int, float, string, boolean, image, kernel and they are mapped to int, float, String, boolean, BufferedImage, and Kernel of Java data types.

5.3 Built-in Functions

Matlip provides some useful built-in functions for image processing, such as imload, imsave, imshow, getheight, getwidth, etc. These functions are printed out by java printer to the output java class file (Matlip.java).

5.4 Symbol Table

The symbol tables are used to keep track the pair of variable/function name and its type. Maplip implements three symbol tables for functions, global variables, and local variables by the data structure of name map and list. Matlip is static scope; when accessing the value of a variable, local symbol is searched first; then global symbol table.

6. Test Plan

6.1 Transforming Matlip code to Java

Matlip Source Program (Flip the image vertically)	Target Java Program

function image ret = flip(image im)
int height;
int width;
int i;
int j;
height = getheight(im);
width = getwidth(im);
ret=imnew(width,height,"RGB");
for j=0:height-1
    for i=0:width-1
        ret[i,height-j-1,"rgb"]=im[i,j,"rgb"];
    end
end
end
function = main()
   image x;
   image y;
   x=imread("./rabbit.jpg");
   imshow(x);
   y=flip(x);
   imshow(y);
 end
	public static void main(String[] args)
	{
	BufferedImage x = imnew(100, 100, "RGB");
	BufferedImage y = imnew(100, 100, "RGB");
	x = imread("./rabbit.jpg");
	imshow(x);
	y = flip(x);
	imshow(y);
	}
	public static BufferedImage flip(BufferedImage im)
	{
	BufferedImage ret = imnew(100, 100, "RGB");
	int height = 0;
	int width = 0;
	int i = 0;
	int j = 0;
	height = getHeight(im);
	width = getWidth(im);
	ret = imnew(width, height, "RGB");
	for (j = 0 ; j <= height - 1 ; j++){
	for (i = 0 ; i <= width - 1 ; i++){
	setImagePixel(ret, i, height - j - 1, "rgb", getImagePixel(im, i, j, "rgb"));
	}
	}
	return ret;
	}

Matlip Source Program (Flip the image vertically)	Target Java Program

function = main()
image x;
image y;
kernel k;
k=[0.0,-1.0,0.0;-1.0,5.0,-1.0;0.0,-1.0,0.0];
x=imread("./rabit.gif");
imshow(x);
y=x@k;
imshow(y);
end
	public static void main(String[] args)
	{
	BufferedImage x = imnew(100, 100, "RGB");
	BufferedImage y = imnew(100, 100, "RGB");
	Kernel k = kernelinit();
	k = new Kernel (3, 3, new float[]{(float)0., -((float)1.), (float)0., -((float)1.), (float)5., -((float)1.), (float)0., - ((float)1.), (float)0.});
	x = imread("./rabit.gif");
	imshow(x);

	y = new ConvolveOp(k,
	ConvolveOp.EDGE_NO_OP,null).filter(x, null);
	imshow(y);
	}

6.2 Test Suite

Myshell.sh, written in shell script, is the automation testing suite to test our translator. It takes the file name ending with .mp as input and check the syntax of the program. If no syntax errors are found, it will translate the Matlip source code into java, which will then be compiled and executed. At last, it will compare the output with our expected answers and write the results into matlip-test.log. On the other hand, if there is syntax error in the test program, all the subsequent actions will be stopped and the error messages will be written to the matlip-test.log.

6.3 Test Case

Grammar		

`for` *`variable`* `=` *`expression1`* `:` *`constant`* `:` *`expression2`*
   `statements`
`end`		

Equivalence Class		**Boundary Values**
`Valid constant`	`Constant is negative`	`-1`
	`Constant is positive`	`1`
	`Constant is 0`	`0`

`Invalid constant`	`Constant is identifier`	`int x;`
		`x`

	`Constant is expression`	`int i;`
		`i+1`

Grammar		

`Function declaration`
`function` *`return_value`* `=` *`function_name`*`(`*`argument_list`*`)`
      `statements`
`end`
`Function Call`
`result` `=` *`function_name`*`(`*`arguments`*`);/`*`function_name`*`(`*`arguments`*`);`		

Equivalence Class		**Boundary Values**

`function declaration and function call mismatch`	`The number of argument in the function declaration is greater  than`	`function = test (int x)`
		`end`
		`function = main ()`
		`   test();`
		`end`

	that of the function call	

	The number of argument in the function declaration is smaller than that of the function call	Function = test ()
		end
		function = main ()
		   int x;
		   test(x);
		end

	Type of the argument in the function declaration is different from that of the function call	function = test (int x)
		end
		function = main ()
		float y;
		   test(y);
		end

	function call without function declaration	function = main ()
		   test();
		end

function declaration and function call match	With argument	function = test (int x)
		end
		function = main ()
		   int x;
		   test(x);
		end

	Without argument	function = test ()
		end
		function = main ()
		   test();
		end

	Recursion	function int m= test(int x)
		  if (x==0)
		      m=0;
		  else
		     m=test(x);
		   end
		end
		function = main ()
		   test(5);
		end

<table>
<tr><th colspan="3">Grammar</th></tr>
<tr><td>if bool_type
   <i>statements</i>
end</td><td>if bool_type
   <i>statements</i>
else/elseif bool_type
   <i>statements</i>
end</td><td>if bool_type
   <i>statements</i>
elseif bool_type
   <i>statements</i>
else
   <i>statements</i>
end</td></tr>
<tr><th colspan="2">Equivalence Class</th><th>Boundary Values</th></tr>
<tr><td rowspan="2">the parameter of if or elseif is not of bool type</td><td>The parameter of if or elseif is of int type</td><td>int x;
x=3;
if 3
elseif x;
end</td></tr>
<tr><td>The parameter of if</td><td>string x;</td></tr>
</table>

	or elseif is of string type	if x
		end

	The parameter of if or elseif is of float type	if 3.0
		end

statement is if statement		if x==3
		   if y==4
		      x=x+1;
		   else
		      x=x-1;
		   end
		else
		   if x==5
		      x=x-1;
		   end
		end

Grammar		

While bool_type
   statements
end		

Equivalence Class		**Boundary Values**

the parameter of while is not of bool type	The parameter of while is of int type	while 3
		end

	The parameter of while is of string type	string x;
		while x
		end

	The parameter of while is of float type	while 3.0
		end

Grammar		
imread(string path); is of image type		
Equivalence Class		**Boundary Values**

Invalid input	The parameter is of int type	int x;
		imread(x);

		imread(3);

	The parameter is of float type	float x;
		imread(x);

		imread(3.0);

	The parameter is of image type	image x;
		imread(x);

Valid input	The parameter is of string type	imread(./rabbit.jpg);

		string k;
		k=”./rabbit.jpg”;
		imread(k);

Grammar

imsave(A, B); A is of image type ,B is of string type		
Equivalence Class		Boundary Values

Invalid input	A is of int type, B is of string type	int x;
		string y;
		imsave(x,y);

	A is of string type, B is of string type	string x;
		string y;
		imsave(x,y);

	A is of kernel type, B is of string type	kernel x;
		string y;
		imsave(x,y);

	A is of image type, B is of int type	image x;
		int y;
		imsave(x,y);

	A is of image type, B is of kernel type	image x;
		kernel y;
		imsave(x,y);

	A is of image type, B is of float type	image x;
		float y;
		imsave(x,y);

Valid input	The parameter is of string type	image x;
		string k;
		k=”./rabbit.jpg”;
		imsave(x,k);

Grammar		
newImage(int width, int height, String type)		
Equivalence Class		Boundary Values

Invalid input	width and height are of float type, TYPE is of string type	float x;
		float y;
		string z;
		imsave(x,y,z);

	width and height are of string type, TYPE is of string type	string x;
		string y;
		string z;
		imsave(x,y,z);

	width and height are of kernel type, TYPE is of string type	kernel x;
		kernel y;
		string z
		imsave(x,y,z);

	width and height are of int type, TYPE is of image type	int x;
		int y;
		image z
		imsave(x,y,z);

	width and height are of int type, TYPE is of kernel type	int x;
		int y;
		kernel z;
		imsave(x,y,z);

	width and height are of int type, TYPE is of float type	int x;
		int y;
		float z;
		imsave(x,y,z);

Valid input	width and height	int x;

	are of int type, TYPE is of string type	int y;
		string z;
		z=”./rabbit.jpg”;
		imsave(x,y,z);

SCOPE		
Equivalence Class	**Test case**	**Expected result**

Declare a global variable and call it in a function	int x;
	function = main()
	x=1;
	print(x);
	end
		1

Declare a local variable in a function and call it in other function	function = test()
	int x;
	end
	function = main()
	x=1;
	print(x);
	end
		Error

Static Scoping	int x;
	function int m= test()
	m=x;
	end
	function int m= test2()
	int x;
	x=1;
	m=test();
	end
	function = main()
	print(test2());
	end
		0

IMAGE ASSIGNMENT AND OPERATION		
Equivalence Class		**Boundary value**

one side of the operator is of image type and assign the result to image type	Left hand side is image and right hand side is integer	image x;
		int i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	Left hand side is image and right hand side is string	image x;
		string i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	Left hand side is image and right hand side is	image x;
		kernel i;
		x=x+i;

	kernel	x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	Left hand side is image and right hand side is float	image x;
		float i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	right hand side is image and left hand side is integer	image x;
		int i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	right hand side is image and left hand side is string	image x;
		string i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	right hand side is image and left hand side is kernel	image x;
		kernel i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	right hand side is image and left hand side is float	image x;
		float i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

two sides of the operator is of image type and assign the result to image type	The dimension of both sides do not match	image x;
		image i;
		i=imread(./rabbit.jpg);
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	The dimension of	image x;

	both sides match	image i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

KERNEL ASSIGNMENT AND OPERATION		
Equivalence Class		**Boundary value**

one side of the operator is of kernel type and assign the result to kernel type	Left hand side is kernel and right hand side is integer	kernel x;
		int i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	Left hand side is kernel and right hand side is string	kernel x;
		string i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	Left hand side is kernel and right hand side is kernel	kernel x;
		kernel i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	Left hand side is kernel and right hand side is float	kernel x;
		float i;
		x=x+i;
		x=x-i;
		x=x*i;
		x=x/i;
		x=x mod i;
		x=x^i;

	right hand side is kernel and left hand side is integer	kernel x;
		int i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	right hand side is kernel and left hand side is	kernel x;
		string i;
		x=i+x;

	string	x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	right hand side is kernel and left hand side is kernel	kernel x;
		kernel i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	right hand side is kernel and left hand side is float	kernel x;
		float i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

two sides of the operator is of kernel type and assign the result to kernel type	The dimension of both sides do not match	kernel x;
		kernel i;
		i=kernelnew(10,10);
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

	The dimension of both sides match	kernel x;
		kernel i;
		x=i+x;
		x=i-x;
		x=i*x;
		x=i/x;
		x=i mod x;
		x=i^x;

Free-Form Function Declaration		
Equivalence Class		**Boundary Values**

Function declared before being used		function =test()
		print(x);
		end
		function main()
		int x;
		test();
		end

Function declared after being used		function =test2()
		print(x);
		end
		function main()
		int x;
		test();

		end
		function =test()
		x=x+1;
		test2();
		end

Namesapce		
Equivalence Class		**Boundary Values**

Global variable name is the same with global function name		int x;
		function int m=x()
		end

Local variable name is the same with global function name	Local variable is the parameter of the global function	function int x=x() end

	Local variable is inside the scope of the global function	function int m=x()
		int x;
		m=x;
		end

	Local variable is outside of the scope of the global function	function int m=test()
		end
		function = main()
		int test;
		end

Local variable name is the same with global variable name	Local variable is the parameter of the function	int x;
		function int x=test()
		end

	Local variable is inside one function	int x;
		function int m=test()
		int x;
		end

Two global variable with the same name		int x; int x;

Two global function with the same name		function = test()
		end
		function = test()
		end

7. LESSONS LEARNED

PoHsu Yeh

From this project, I learned the importance of testing and version control system. Slight chances might break the whole system; therefore I suggest to do the regression test immediately whenever the changes are made. Furthermore, version control is strongly recommended. Plan early, start early, and try to minimize the changes when the project is about to finish because it would require lots of work to fix the bugs.

Pin-Chin Huang

We should have elected a team lead when the project started. When it comes to team projects, dictatorship in indeed more efficient. We should also have divided project responsibilities among team members early, although it might be hard when the tasks were not yet clearly defined. For the implementation part, I should have planned ahead and thought through before starting to write code. For instance, I first thought the javaprinter would not need a symbol table. But when I started to code the image and kernel part, it became clear that I would also need to evaluate the type of each expression, and adding symbol tables to my code late in time was rather a hassle. For the testing part, it also became clear that automatic regression testing is absolutely necessary for a compiler project since even a small change can break many features. Also, given the vast number of possible test cases for a new programming language, black-box testing might not always work: the code authors should have been part of the test case planning. This would have saved me from fixing the consecutive image assignment bug and the global/local symbol table problems just a few hours from the demo.

SHIH-HAO, LIAO

First, I think it is better not to use OCAML to develop our language. Although functional programming requires fewer lines of code to finish the project, it is difficult to divide the core part of the project into many subsets. Every time when I try to modify some part of the walker, it is impossible for me to integrate the code written by others. At last, I stop doing the walker but focused on the testing of our program. It is better that I can write the testing report and test our language at the same time so that it will be easier for me to define the equivalence class which will make the testing more efficient and organized. Besides, I think the best way to do the testing in our project is not black box testing. There are many times where white box testing can identify an error faster than black box testing.

SHARIAR

I have learned a lot in how build a translator capable of syntax checking.

APPENDIX A

File Name	Author
Makefile	Shariar Pin-Chin Huang Shih-Hao, Liao
Ast.mli	Shariar Pin-Chin Huang Shih-Hao, Liao
Parser.mly	Shariar Pin-Chin Huang Shih-Hao, Liao
Scanner.mll	Shariar Pin-Chin Huang Shih-Hao, Liao
Matlip.ml	Shariar Pin-Chin Huang Shih-Hao, Liao
Walker.ml	Shariar
Javaprinter.ml	Pin-Chin Huang
Myshell.sh	Shih-Hao, Liao PoHsu Yeh

All the file in Matlip-java	PoHsu Yeh
All the file in tests	Shih-Hao, Liao (Primary) PoHsu Yeh
MATLIP.ppt	Shih-Hao, Liao

Appendix B.

Makefile

```
#OBJS = parser.cmo scanner.cmo printer.cmo microc.cmo
OBJS = parser.cmo scanner.cmo walker.cmo javaprinter.cmo matlip.cmo

TESTS = \
arith1 \
arith2 \
fib \
for1 \
func1 \
gcd \
global1 \
hello \
if1 \
if2 \
if3 \
if4 \
ops1 \
var1 \
while1

TARFILES = Makefile myshell.sh scanner.mll parser.mly \
      ast.mli walker.ml matlip.ml javaprinter.ml\
      $(TESTS:%=tests/test-%.mc) \
      $(TESTS:%=tests/test-%.out)

#TARFILES = Makefile testall.sh scanner.mll parser.mly \
#     ast.mli printJava.ml microc.ml \
#     $(TESTS:%=tests/test-%.mc) \
#     $(TESTS:%=tests/test-%.out)

matlip : $(OBJS)
      ocamlc -o matlip $(OBJS)

.PHONY : test
test : matlip myshell.sh
      ./myshell.sh

scanner.ml : scanner.mll
      ocamllex scanner.mll

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

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

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

microc.tar.gz : $(TARFILES)
```

```
      cd .. && tar czf microc/microc.tar.gz $(TARFILES:%=microc/%)

.PHONY : clean
clean :
      rm -f matlip parser.ml parser.mli scanner.ml matlip-test.log
*.cmo *.cmi *.class Matlip.java

# Generated by ocamldep *.ml *.mli
matlip.cmo: scanner.cmo parser.cmi
matlip.cmx: scanner.cmx parser.cmx
parser.cmo: ast.cmi parser.cmi
parser.cmx: ast.cmi parser.cmi
#printer.cmo: ast.cmi
#printer.cmx: ast.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmi
```

ast.mli

```
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater
| Geq | And | Or | Convolve | Power

type expr =
    Literal of int
  | Floatlit of float
  | Id of string
  | Datatype of string
  | String of string
  | Binop of expr * op * expr
  | Assign of string * expr
  | AssignPixel of string * expr * expr * expr * expr
  | AssignKernel of string * expr * expr * expr
  | Call of string * expr list
  | Uminus of expr
  | Not of expr
  | ImageAccess of string * expr * expr * expr
  | KernelAccess of string * expr * expr
  | Imnew of expr * expr * string
  | Kernelnew of expr * expr
  | Bool of string

type for_expr =
    Assigna of string * expr * expr
  | Assignb of string * expr * expr * expr

type stmt =
    Block of stmt list
  | Expr of expr
  | KernelInit of string * expr list list
  | If of expr * stmt * stmt
  | Ifelseif of expr * stmt * elseif list * stmt
  | For of for_expr * stmt
  | While of expr * stmt
and elseif = {
    elseif_expr : expr;
    elseif_stmt : stmt;
  }
```

```
type var_decl = {
    varname : string;
    vartype : string;
  }

type func_decl = {
    fname : string;
    rettype : string;
    retname : string;
    formals : var_decl list;
    locals : var_decl list;
    body : stmt list;
  }

type program = var_decl list * func_decl list
```

Scanner.mll

```
{
    open Parser

    let incr_lineno lexbuf =
        let pos = lexbuf.Lexing.lex_curr_p in
            lexbuf.Lexing.lex_curr_p <- { pos with
                Lexing.pos_lnum = pos.Lexing.pos_lnum + 1;
                Lexing.pos_bol = pos.Lexing.pos_cnum;
            }
}

rule token = parse
  [' ' '\t'] { token lexbuf }
| ['\r' '\n']  { incr_lineno lexbuf; token lexbuf }
| '#'      { comment lexbuf }
| '"'      { string_type "" lexbuf}
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| '@'      { CONVOLVE }
| '^'      { POWER }
| ';'      { SEMI }
| ','      { COMMA }
| ':'      { COLON }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "and"    { AND }
| "or"     { OR }
| "not"    { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "elseif" { ELSEIF }
| "for"    { FOR }
| "while"  { WHILE }
| "int"  { DATATYPE("int") }
| "float"  { DATATYPE("float") }
| "string" { DATATYPE("string") }
| "image" { DATATYPE("image") }
| "imnew" { NEWIMAGE }
| "kernel" { DATATYPE("kernel") }
| "kernelnew" { NEWKERNEL }
| "boolean"  { DATATYPE("boolean") }
| "end"    { END }
| "function" { FUNCTION }
| "true"  as lxm { BOOL(lxm) }
| "false"  as lxm { BOOL(lxm) }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['0'-'9']+'.'['0'-'9']+(['E''e']['+''-']?['0'-'9']+)? as lxm
{ FLOATLIT(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char {
            let pos = lexbuf.Lexing.lex_curr_p in
            raise (Failure("Illegal character: " ^ Char.escaped char
            ^ " in line #" ^ (string_of_int pos.Lexing.pos_lnum))) }

and comment = parse
  "\n" { incr_lineno lexbuf; token lexbuf }
| _    { comment lexbuf }

and string_type str = parse
   '"' { if (String.length str) > 0 then
            if str.[(String.length str)-1] = '\\' then
                string_type (str^(Lexing.lexeme lexbuf)) lexbuf
            else STRING(str)
         else STRING(str) }
|  "\n" {   let pos = lexbuf.Lexing.lex_curr_p in
            raise (Failure("Unclosed string literal, found beginning
of"
            ^ " a new line without closure of string; "
            ^ " in line #" ^ (string_of_int pos.Lexing.pos_lnum))) }
|  _ { string_type (str^(Lexing.lexeme lexbuf)) lexbuf }
```

Paser.mly

```
%{  open Ast
    open Lexing
    let parse_error msg =
        let start_pos = Parsing.rhs_start_pos 1 in
                                let lineNo = start_pos.pos_lnum in
                                print_endline (msg ^ " in line #" ^
string_of_int lineNo)
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA COLON LBRACKET RBRACKET
%token PLUS MINUS TIMES DIVIDE ASSIGN AND OR NOT CONVOLVE POWER
%token EQ NEQ LT LEQ GT GEQ
%token IF ELSE ELSEIF FOR WHILE END FUNCTION NEWIMAGE NEWKERNEL
```

```
%token <int> LITERAL
%token <float> FLOATLIT
%token <string> ID
%token <string> DATATYPE
%token <string> STRING
%token <string> BOOL
%token EOF

%nonassoc NOELSE
%nonassoc ELSE

%nonassoc NOACTUALS
%nonassoc LPAREN

%nonassoc NOMINUS

%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left CONVOLVE
%left POWER
%right NOT
%nonassoc UMINUS

%start program
%type <Ast.program> program

%%

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

fdecl:
   FUNCTION DATATYPE ID ASSIGN ID LPAREN formals_opt RPAREN
vdecl_list stmt_list END
     { { fname = $5;
     rettype = $2;
     retname = $3;
       formals = $7;
       locals = List.rev $9;
       body = List.rev $10 } }
  |  FUNCTION ASSIGN ID LPAREN formals_opt RPAREN vdecl_list
stmt_list END
     { { fname = $3;
     rettype = "void";
     retname = "";
       formals = $5;
       locals = List.rev $7;
       body = List.rev $8 } }

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

formal_list:
    param_decl          { [$1] }
  | formal_list COMMA param_decl  { $3 :: $1 }

param_decl:
   DATATYPE ID
     { { varname = $2;
     vartype = $1 } }

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

vdecl:
   DATATYPE ID SEMI
     { { varname = $2;
     vartype = $1 } }


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

stmt:
    expr SEMI { Expr($1) }
  | error SEMI { Expr(Datatype("parseerror")) }
  | IF expr stmt_list END
        { If($2, Block(List.rev $3), Block([])) }
  | IF expr stmt_list ELSE stmt_list END
        { If($2, Block(List.rev $3), Block(List.rev $5)) }
  | IF expr stmt_list elseif_list END
        { Ifelseif($2, Block(List.rev $3), List.rev $4, Block([])) }
  | IF expr stmt_list elseif_list ELSE stmt_list END
        { Ifelseif($2, Block(List.rev $3), List.rev $4,
Block(List.rev $6)) }
  | FOR for_expr stmt_list END
        { For($2, Block(List.rev $3)) }
  | WHILE expr stmt_list END { While($2, Block(List.rev $3)) }
  | ID ASSIGN LBRACKET array_list RBRACKET SEMI
        { KernelInit($1, List.rev $4) }

array_list:
    expr          { [[$1]] }
  | array_list COMMA expr  {
        if (List.length $1) = 1 then
            [$3 :: List.hd $1]
        else
            ($3 :: List.hd $1) :: List.tl $1 }
  | array_list SEMI expr  { [$3] :: $1 }


/*
  expr_opt:
                 { Noexpr }
  | expr         { $1 }
*/
expr:
    LITERAL          { Literal($1) }
  | FLOATLIT         { Floatlit($1) }
  | BOOL             { Bool($1) }
  | ID %prec NOACTUALS     { Id($1) }
  | STRING           { String($1) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr AND   expr  { Binop($1, And,   $3) }
  | expr OR  expr    { Binop($1, Or,   $3) }
  | expr POWER  expr { Binop($1, Power,  $3) }
  | NOT expr         { Not($2) }
  | MINUS expr %prec UMINUS    { Uminus($2) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater,  $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr CONVOLVE expr { Binop($1, Convolve, $3) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID LBRACKET expr COMMA expr COMMA expr RBRACKET ASSIGN expr
        {AssignPixel($1, $3, $5, $7, $10)}
  | ID LBRACKET expr COMMA expr RBRACKET ASSIGN expr
        {AssignKernel($1, $3, $5, $8)}
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | ID LBRACKET expr COMMA expr COMMA expr RBRACKET
        { ImageAccess($1, $3, $5, $7) }
  | ID LBRACKET expr COMMA expr RBRACKET
        { KernelAccess($1, $3, $5) }
  | NEWIMAGE LPAREN expr COMMA expr COMMA STRING RPAREN
        { Imnew($3, $5, $7) }
  | NEWKERNEL LPAREN expr COMMA expr RPAREN
        { Kernelnew($3, $5) }

for_expr:
    ID ASSIGN expr COLON expr %prec NOMINUS {Assigna($1, $3, $5)}
  | ID ASSIGN expr COLON expr COLON expr %prec NOMINUS
    {Assignb($1, $3, $5, $7)}

elseif_list:
    elseif_clause             { [$1] }
  | elseif_list elseif_clause { $2 :: $1 }

elseif_clause:
    ELSEIF expr stmt_list
     { {elseif_expr = $2;
        elseif_stmt = Block(List.rev $3)} }

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

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

walker.mll

```
open Ast

module NameMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end)

let string_of_program (vars, funcs) =
  (* Put function declarations in a symbol table *)
  let func_decls = List.fold_left
        (fun funcs fdecl ->
            if NameMap.mem fdecl.fname funcs then
                raise (Failure ("Function: '" ^ fdecl.fname ^
                    "' has already been defined."))
            else
                NameMap.add fdecl.fname fdecl funcs) NameMap.empty
funcs
    in
    let rec string_of_fdecl globals fdecl actuals =

    let rec string_of_expr globals locals fname = function
        Literal(l) -> { varname = string_of_int l;vartype = "int" }
      | Floatlit(f) -> { varname = string_of_float f;vartype =
"float" }
      | String(s) -> { varname = s;vartype = "string" }
      | Bool(s) -> { varname = s;vartype = "boolean" }
      | Id(s) ->
            if NameMap.mem s locals then
              NameMap.find s locals
            else if NameMap.mem s globals then
              NameMap.find s globals
          else raise (Failure ("Undeclared identifier found in
expression: '"
              ^ s ^ "' in function: '" ^ fname ^ "'"))
      | Datatype(t) ->
            if t = "parseerror" then
                raise ((Failure ("Syntax error")))
            else { varname = "null" ;vartype = t }
      | Uminus(e) -> let v = string_of_expr globals locals fname e in
            if v.vartype = "int" || v.vartype = "float" then
                v
            else
                raise (Failure ("unary - operator can only be applied
to int or float expressions"
                                ^ ", but here used with '" ^
v.varname ^ "', type: "
                                ^ v.vartype ^ " in function: '" ^
fname ^ "'"))
      | Not(e) -> let v = string_of_expr globals locals fname e in
            if v.vartype = "boolean" then
                v
            else
                raise (Failure ("not operator can only be applied to
boolean expression"
                                ^ ", but here used with '" ^
v.varname ^ "', type: "
                                ^ v.vartype ^ " in function: '" ^
fname ^ "'"))
      | Binop(e1, o, e2) ->
          let v1 = string_of_expr globals locals fname e1 in
          (match o with
             Add ->
             let v2 = string_of_expr globals locals fname e2 in
               if v1.vartype = "string" then
```

```
                 if v2.vartype = "image" || v2.vartype = "kernel"
then
                   raise (Failure ("Cannot concatenate " ^ v2.vartype
^ " type with string type " ^
                                   "in function: '" ^ fname ^ "'"))
                 else v1
               else if v2.vartype = "string" then
                 if v1.vartype = "image" || v1.vartype = "kernel"
then
                   raise (Failure ("Cannot concatenate " ^ v1.vartype
^ " type with string type " ^
                                   "in function: '" ^ fname ^ "'"))
                 else v2
               else if v1.vartype = "image" && v2.vartype = "int"
then
                      v1
               (*else if v2.vartype = "image" && v1.vartype = "int"
then
                      v2*)
               else if v1.vartype = "kernel" && v2.vartype = "float"
then
                      v1
               else if v1.vartype = "boolean" || v2.vartype =
"boolean" then
                 raise (Failure ("+ operator cannot be applied to
boolean operands"
                                  ^ ", but here used with '" ^
v1.varname ^ "', type: "
                                  ^ v1.vartype ^ " and '" ^
v2.varname ^ "', type: "
                                  ^ v2.vartype ^ " in function: '" ^
fname ^ "'"))
               else if v1.vartype <> v2.vartype then
                 raise (Failure ("Type mismatch in binary operation
between: '" ^
                                  v1.varname ^ "', type: " ^
v1.vartype ^ " and '" ^ v2.varname
                                  ^ "', type: " ^ v2.vartype ^ " in
function: '" ^ fname
                                  ^ "'"))
               else v1
         | Sub ->
             let v2 = string_of_expr globals locals fname e2 in
                if v1.vartype = "string" || v2.vartype = "string"
then
                  raise (Failure ("Operators '-' " ^
                                   "cannot be applied to a string, in
function: '" ^ fname ^ "'"))
                else if v1.vartype = "image" && v2.vartype = "int"
then
                      v1
                (*else if v2.vartype = "image" && v1.vartype = "int"
then
                      v2*)
                else if v1.vartype = "kernel" && v2.vartype = "float"
then
                      v1
               else if v1.vartype = "boolean" || v2.vartype =
"boolean" then
                 raise (Failure ("- operator cannot be applied to
boolean operands"
```

```
                                  ^ ", but here used with '" ^
v1.varname ^ "', type: "
                                  ^ v1.vartype ^ " and '" ^
v2.varname ^ "', type: "
                                  ^ v2.vartype ^ " in function: '" ^
fname ^ "'"))
                else if v1.vartype <> v2.vartype then
                  raise (Failure ("Type mismatch in binary operation
between: '" ^
                                   v1.varname ^ "', type: " ^
v1.vartype ^ " and '" ^ v2.varname
                                   ^ "', type: " ^ v2.vartype ^ " in
function: '" ^ fname
                                   ^ "'"))
                else v1
           | Equal | Neq ->
             let v2 = string_of_expr globals locals fname e2 in
                if v1.vartype <> v2.vartype then
                  raise (Failure ("Type mismatch in binary operation
between: '" ^
                                   v1.varname ^ "', type: " ^
v1.vartype ^ " and '" ^ v2.varname
                                   ^ "', type: " ^ v2.vartype ^ " in
function: '" ^ fname
                                   ^ "'"))
                else { varname = "null"; vartype = "boolean" }
         | Mult | Div ->
             let v2 = string_of_expr globals locals fname e2 in
                if v1.vartype = "string" || v2.vartype = "string"
then
                  raise (Failure ("Operators '*', '/' " ^
                                  "cannot be applied to a string, in
function: '" ^ fname ^ "'"))
                else if v1.vartype = "image" && v2.vartype = "int"
then
                    v1
                (*else if v2.vartype = "image" && v1.vartype = "int"
then
                    v2*)
                else if v1.vartype = "kernel" && v2.vartype = "float"
then
                    v1
                else if v1.vartype = "boolean" || v2.vartype =
"boolean" then
                 raise (Failure ("/, * operators cannot be applied to
boolean operands"
                                  ^ ", but here used with '" ^
v1.varname ^ "', type: "
                                  ^ v1.vartype ^ " and '" ^
v2.varname ^ "', type: "
                                  ^ v2.vartype ^ " in function: '" ^
fname ^ "'"))
                else if v1.vartype <> v2.vartype then
                  raise (Failure ("Type mismatch in binary operation
between: '" ^
                                   v1.varname ^ "', type: " ^
v1.vartype ^ " and '" ^ v2.varname
                                   ^ "', type: " ^ v2.vartype ^ " in
function: '" ^ fname
                                   ^ "'"))
                else v1
         | Less | Leq | Greater | Geq ->
             let v2 = string_of_expr globals locals fname e2 in
                if v1.vartype = "string" || v2.vartype = "string"
then
                  raise (Failure ("Operators '<', '<=', '>', '>=' " ^
                                  "cannot be applied to a string, in
function: '" ^ fname ^ "'"))
               else if v1.vartype = "boolean" || v2.vartype =
"boolean" then
                 raise (Failure ("<,<=,>,>= operators cannot be
applied to boolean operands"
                                  ^ ", but here used with '" ^
v1.varname ^ "', type: "
                                  ^ v1.vartype ^ " and '" ^
v2.varname ^ "', type: "
                                  ^ v2.vartype ^ " in function: '" ^
fname ^ "'"))
                else if v1.vartype <> v2.vartype then
                  raise (Failure ("Type mismatch in binary operation
between: '" ^
                                   v1.varname ^ "', type: " ^
v1.vartype ^ " and '" ^ v2.varname
                                   ^ "', type: " ^ v2.vartype ^ " in
function: '" ^ fname
                                   ^ "'"))
                else { varname = "null"; vartype = "boolean" }
         | And | Or ->
             let v2 = string_of_expr globals locals fname e2 in
                if v1.vartype = "boolean" && v2.vartype = "boolean"
then
                    v1
                else
                    raise (Failure ("and, or operators can only be
applied to boolean expressions"
                                  ^ ", but here used with '" ^
v1.varname ^ "', type: "
                                  ^ v1.vartype ^ " and '" ^
v2.varname ^ "', type: "
                                  ^ v2.vartype ^ " in function: '" ^
fname ^ "'"))
          | Convolve ->
            let v2 = string_of_expr globals locals fname e2 in
            if v1.vartype = "image" && v2.vartype = "kernel" then
                v1
            else
                  raise (Failure ("@ operator needs image and kernel
as operands,"
                  ^ " where image is the left operand, and kernel is
the right operand,"
                  ^ " but here got '" ^ v1.vartype ^ ": " ^
v1.varname ^ "' and '"
                  ^ v2.vartype ^ ": " ^ v2.varname ^ "' in function:
'" ^ fname ^ "'"))
          | Power ->
            let v2 = string_of_expr globals locals fname e2 in
            if v1.vartype = "int" && v2.vartype = "int" then
                v1
            else if v1.vartype = "float" && v2.vartype = "float" then
                v1
            else
```

```
                  raise (Failure ("^ operator needs both operands as
int or float,"
                  ^ " but here got '" ^ v1.vartype ^ ": " ^
v1.varname ^ "' and '"
                  ^ v2.vartype ^ ": " ^ v2.varname ^ "' in function:
'" ^ fname ^ "'")))
      | Assign(v, e) ->
            if NameMap.mem v locals then
                let var1 = NameMap.find v locals in
                let var2 = string_of_expr globals locals fname e in
              if var1.vartype <> var2.vartype then
                  raise (Failure ("Type mismatch in assignment
operation between: '"
                      ^ var1.varname ^ "', type: " ^ var1.vartype ^ "
and '"
                      ^ var2.varname ^ "', type: " ^ var2.vartype
                      ^ " in function: '" ^ fname ^ "'"))
              else var1
        else if NameMap.mem v globals then
              let var1 = NameMap.find v globals in
                let var2 = string_of_expr globals locals fname e in
              if var1.vartype <> var2.vartype then
                  raise (Failure ("Type mismatch in assignment
operation between: '"
                      ^ var1.varname ^ "', type: " ^ var1.vartype ^ "
and '"
                      ^ var2.varname ^ "', type: " ^ var2.vartype
                      ^ " in function: '" ^ fname ^ "'"))
              else var1
        else raise (Failure ("Undeclared identifier found in
assignment: '"
              ^ v ^ "' in function: '" ^ fname ^ "'"))
      | AssignPixel(id, row, col, channel, value) ->
            let varchannel = string_of_expr globals locals fname
channel in
            if varchannel.vartype <> "string" then
                raise (Failure ("The channel in pixel assignment must
be of type string"
                    ^ ", but here used with: '" ^ varchannel.vartype
^ "' in function: '"
                    ^ fname ^ "'"))
            else
                let varvalue = string_of_expr globals locals fname
value in
                let varRow = string_of_expr globals locals fname row
in
                let varCol = string_of_expr globals locals fname col
in
                if  varvalue.vartype <> "int" then
                    raise (Failure ("Type mismatch in image pixel
assignment."
                        ^ " The value must be of type int but here
used with type '"
                        ^ varvalue.vartype ^ "' in function: '" ^
fname ^ "'"))
                else
                    if NameMap.mem id locals then
                        let varId = NameMap.find id locals in
                        if varId.vartype <> "image" then
                            raise (Failure ("Expecting image type but
got '" ^ varId.vartype ^
```

```
                                "' for variable: '" ^ varId.varname ^
" in function: '" ^ fname ^ "'"))
                        else if varRow.vartype <> varCol.vartype ||
varRow.vartype <> "int" then
                            raise (Failure ("Type mismatch in image
access. Both row and column"
                                ^ " accessors must be of type int; in
function: '" ^ fname ^ "'"))
                        else { varname = "null";vartype = "int" }
                    else if NameMap.mem id globals then
                        let varId = NameMap.find id globals in
                        if varId.vartype <> "image" then
                        raise (Failure ("Expecting image type but got
'" ^ varId.vartype ^
                            "' for variable: '" ^ varId.varname ^ "
in function: '" ^ fname ^ "'"))
                        else if varRow.vartype <> varCol.vartype &&
varRow.vartype <> "int" then
                            raise (Failure ("Type mismatch in image
access. Both row and column"
                                ^ " accessors must be of type int; in
function: '" ^ fname ^ "'"))
                        else { varname = "null";vartype = "int" }
                    else raise (Failure ("Undeclared identifier found
in assignment: '"
                          ^ id ^ "' in function: '" ^ fname ^ "'"))
      | AssignKernel(id, row, col, value) ->
            let varvalue = string_of_expr globals locals fname value
in
            let varRow = string_of_expr globals locals fname row in
            let varCol = string_of_expr globals locals fname col in
            if  varvalue.vartype <> "float" then
                raise (Failure ("Type mismatch in kernel value
assignment."
                    ^ " The value must be of type float but here used
with type '"
                    ^ varvalue.vartype ^ "' in function: '" ^ fname ^
"'"))
            else
                if NameMap.mem id locals then
                    let varId = NameMap.find id locals in
                    if varId.vartype <> "kernel" then
                        raise (Failure ("Expecting kernel type but
got '" ^ varId.vartype ^
                            "' for variable: '" ^ varId.varname ^ "
in function: '" ^ fname ^ "'"))
                    else if varRow.vartype <> varCol.vartype ||
varRow.vartype <> "int" then
                        raise (Failure ("Type mismatch in kernel
access. Both row and column"
                            ^ " accessors must be of type int; in
function: '" ^ fname ^ "'"))
                    else { varname = "null";vartype = "float" }
                else if NameMap.mem id globals then
                    let varId = NameMap.find id globals in
                    if varId.vartype <> "kernel" then
                    raise (Failure ("Expecting kernel type but got '"
^ varId.vartype ^
                        "' for variable: '" ^ varId.varname ^ " in
function: '" ^ fname ^ "'"))
```

```
                    else if varRow.vartype <> varCol.vartype &&
varRow.vartype <> "int" then
                        raise (Failure ("Type mismatch in kernel
access. Both row and column"
                            ^ " accessors must be of type int; in
function: '" ^ fname ^ "'"))
                    else { varname = "null";vartype = "float" }
              else raise (Failure ("Undeclared identifier found in
assignment: '"
                    ^ id ^ "' in function: '" ^ fname ^ "'"))
      | ImageAccess(id, row, col, channel) ->
            let varchannel = string_of_expr globals locals fname
channel in
            if varchannel.vartype <> "string" then
                raise (Failure ("The channel in pixel access must be
of type string" ^
                      ", but here used with: '" ^ varchannel.vartype
^ "' in function: '" ^ fname ^ "'"))
          else
                let varRow = string_of_expr globals locals fname row
in
                let varCol = string_of_expr globals locals fname col
in
                if NameMap.mem id locals then
                    let varId = NameMap.find id locals in
                    if varId.vartype <> "image" then
                        raise (Failure ("Expecting image type but got
'" ^ varId.vartype ^
                            "' for variable: '" ^ varId.varname ^ "
in function: '" ^ fname ^ "'"))
                    else if varRow.vartype <> varCol.vartype ||
varRow.vartype <> "int" then
                        raise (Failure ("Type mismatch in image
access. Both row and column"
                            ^ " accessors must be of type int; in
function: '" ^ fname ^ "'"))
                    else { varname = "null";vartype = "int" }
              else if NameMap.mem id globals then
                    let varId = NameMap.find id globals in
                    if varId.vartype <> "image" then
                        raise (Failure ("Expecting image type but got
'" ^ varId.vartype ^
                            "' for variable: '" ^ varId.varname ^ "
in function: '" ^ fname ^ "'"))
                    else if varRow.vartype <> varCol.vartype &&
varRow.vartype <> "int" then
                        raise (Failure ("Type mismatch in image
access. Both row and column"
                            ^ " accessors must be of type int; in
function: '" ^ fname ^ "'"))
                    else { varname = "null";vartype = "int" }
              else raise (Failure ("Undeclared identifier found in
function: '" ^ fname ^ "'"))
      | KernelAccess(id, row, col) ->
            let varRow = string_of_expr globals locals fname row in
            let varCol = string_of_expr globals locals fname col in
            if NameMap.mem id locals then
                let varId = NameMap.find id locals in
                if varId.vartype <> "kernel" then
                  raise (Failure ("Expecting kernel type but got '" ^
varId.vartype ^
```

```
                      "' for variable: '" ^ varId.varname ^ " in
function: '" ^ fname ^ "'"))
              else if varRow.vartype <> varCol.vartype ||
varRow.vartype <> "int" then
                  raise (Failure ("Type mismatch in kernel access.
Both row and column"
                      ^ " accessors must be of type int; in function:
'" ^ fname ^ "'"))
              else { varname = "null";vartype = "float" }
          else if NameMap.mem id globals then
                let varId = NameMap.find id globals in
                if varId.vartype <> "kernel" then
                  raise (Failure ("Expecting kernel type but got '" ^
varId.vartype ^
                      "' for variable: '" ^ varId.varname ^ " in
function: '" ^ fname ^ "'"))
              else if varRow.vartype <> varCol.vartype &&
varRow.vartype <> "int" then
                  raise (Failure ("Type mismatch in kernel access.
Both row and column"
                      ^ " accessors must be of type int; in function:
'" ^ fname ^ "'"))
              else { varname = "null";vartype = "float" }
          else raise (Failure ("Undeclared identifier found in
function: '" ^ fname ^ "'"))
      | Imnew(row, col, format) ->
            if format <> "rgb" && format <> "RGB" && format <> "gray"
&& format <> "GRAY"
               && format <> "grey" && format <> "GREY" then
                raise (Failure ("The image format must be either rgb
or gray scale" ^
                      ", but here used with: '" ^ format ^ "' in
function: '" ^ fname ^ "'"))
          else
                let varRow = string_of_expr globals locals fname row
in
                let varCol = string_of_expr globals locals fname col
in
              if varRow.vartype <> varCol.vartype || varRow.vartype
<> "int" then
                  raise (Failure ("Type mismatch in image access.
Both row and column"
                      ^ " accessors must be of type int; in function:
'" ^ fname ^ "'"))
                else
                  { varname = "imnew";vartype = "image" }
      | Kernelnew(row, col) ->
                let varRow = string_of_expr globals locals fname row
in
                let varCol = string_of_expr globals locals fname col
in
              if varRow.vartype <> varCol.vartype || varRow.vartype
<> "int" then
                  raise (Failure ("Type mismatch in image access.
Both row and column"
                      ^ " accessors must be of type int; in function:
'" ^ fname ^ "'"))
                else
                  { varname = "kernelnew";vartype = "kernel" }
      | Call("print", actuals) ->
            let actuals = List.fold_left
```

```
        (fun actuals actual ->
         let v = string_of_expr globals locals fdecl.fname
actual in
           v :: actuals) [] actuals
        in
        if (List.length actuals) <> 1 then
          raise (Failure ("Wrong number of arguments passed
to function 'print'"
            ^ ", called from function: '" ^ fname ^ "'"))
        else
          let param = List.hd actuals in
            if param.vartype <> "image" && param.vartype
<> "kernel" then
              { varname = "print" ; vartype = "void" }
            else
              raise (Failure ("Argument passed to
function 'print' cannot be "
                ^ "image or kernel type, called from
function: '" ^ fname ^ "'"))
   | Call("tostring", actuals) ->
      let actuals = List.fold_left
        (fun actuals actual ->
         let v = string_of_expr globals locals fdecl.fname
actual in
           v :: actuals) [] actuals
        in
        if (List.length actuals) <> 1 then
          raise (Failure ("Wrong number of arguments passed
to function 'tostring'"
            ^ ", called from function: '" ^ fname ^ "'"))
        else
          let param = List.hd actuals in
            if param.vartype <> "image" && param.vartype
<> "kernel" then
              { varname = "tostring" ; vartype =
"string" }
            else
              raise (Failure ("Argument passed to
function 'tostring' cannot be "
                ^ "image or kernel type, called from
function: '" ^ fname ^ "'"))
   | Call("toint", actuals) ->
      let actuals = List.fold_left
        (fun actuals actual ->
         let v = string_of_expr globals locals fdecl.fname
actual in
           v :: actuals) [] actuals
        in
        if (List.length actuals) <> 1 then
          raise (Failure ("function 'toint' takes just one
parameter"
            ^ ", called from function: '" ^ fname ^ "'"))
        else
          let param = List.hd actuals in
            if param.vartype = "float" || param.vartype =
"string" then
              { varname = "toint" ; vartype = "int" }
            else
              raise (Failure ("Argument passed to
function 'toint' can only be "
```

```
                                ^ "either float or string type,
called from function: '" ^ fname ^ "'"))
      | Call("tofloat", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("function 'tofloat' takes just
one parameter"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "int" || param.vartype =
"string" then
                            { varname = "tofloat" ; vartype =
"float" }
                        else
                            raise (Failure ("Argument passed to
function 'tofloat' can only be "
                                ^ "either int or string type, called
from function: '" ^ fname ^ "'"))
      | Call("togray", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("function 'togray' takes just one
parameter"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "image"then
                            { varname = "togray" ; vartype = "image" }
                        else
                            raise (Failure ("Argument passed to
function 'togray' can only be "
                                ^ "image type, called from function:
'" ^ fname ^ "'"))
      | Call("imread", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("Wrong number of arguments passed
to function 'imread'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "string" then
                            { varname = "imread" ; vartype = "image" }
                        else
```

```
                            raise (Failure ("Argument passed to
function 'imread' must be of string type"
                                ^ ", called from function: '" ^ fname
^ "'"))
      | Call("imsave", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 2 then
                    raise (Failure ("Wrong number of arguments passed
to function 'imsave'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param1 = List.nth actuals 1 in
                        if param1.vartype = "image" then
                            let param2 = List.nth actuals 0 in
                            if param2.vartype = "string" then
                                { varname = "imsave" ; vartype =
"void" }
                            else
                                raise (Failure ("Second argument
passed to function 'imsave' must be of"
                                ^ " string type, called from function:
'" ^ fname ^ "'"))
                        else
                            raise (Failure ("First argument passed to
function 'imsave' must be of"
                            ^ " image type, called from function: '"
^ fname ^ "'"))
      | Call("imshow", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("Wrong number of arguments passed
to function 'imshow'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "image" then
                            { varname = "imshow" ; vartype = "void" }
                        else
                            raise (Failure ("Argument passed to
function 'imshow' must be of image type"
                                ^ ", called from function: '" ^ fname
^ "'"))
      | Call("getwidth", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
```

```
                    raise (Failure ("Wrong number of arguments passed
to function 'getwidth'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "image" || param.vartype =
"kernel" then
                            { varname = "getwidth" ; vartype = "int" }
                        else
                            raise (Failure ("Argument passed to
function 'getwidth' must be of either "
                            ^ "image or kernel type, called from
function: '" ^ fname ^ "'"))
      | Call("getheight", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("Wrong number of arguments passed
to function 'getheight'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "image" || param.vartype =
"kernel" then
                            { varname = "getheight" ; vartype =
"int" }
                        else
                            raise (Failure ("Argument passed to
function 'getheight' must be of either "
                            ^ "image or kernel type, called from
function: '" ^ fname ^ "'"))
      | Call("getlength", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("Wrong number of arguments passed
to function 'getlength'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "string" then
                            { varname = "getlength" ; vartype =
"int" }
                        else
                            raise (Failure ("Argument passed to
function 'getlength' must be of "
                            ^ "string type, called from function: '"
^ fname ^ "'"))
      | Call("gettype", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
```

```
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("Wrong number of arguments passed
to function 'gettype'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "image" then
                            { varname = "gettype" ; vartype =
"string" }
                        else
                            raise (Failure ("Argument passed to
function 'gettype' must be of "
                            ^ "image type, called from function: '" ^
fname ^ "'"))
      | Call("mod", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 2 then
                    raise (Failure ("Wrong number of arguments passed
to function 'mod'"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param1 = List.nth actuals 1 in
                    let param2 = List.nth actuals 0 in
                        if param1.vartype = "int" && param2.vartype =
"int" then
                            { varname = "mod" ; vartype = "int" }
                        else if param1.vartype = "float" &&
param2.vartype = "float" then
                            { varname = "mod" ; vartype = "float" }
                        else
                            raise (Failure ("Arguments passed to
function 'mod' must be all of"
                            ^ " either int or float type,"
                            ^ " but here got '" ^ param1.vartype ^ "'
and '"
                              ^ param2.vartype ^ "' called from
function: '" ^ fname ^ "'"))
      | Call("sqrt", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 1 then
                    raise (Failure ("function 'sqrt' takes just one
parameter"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    let param = List.hd actuals in
                        if param.vartype = "float" || param.vartype =
"int" then
                            { varname = "sqrt" ; vartype = "float" }
                        else
```

```
                            raise (Failure ("Argument passed to
function 'sqrt' can only be "
                                ^ "either float or int type, called
from function: '" ^ fname ^ "'"))
      | Call("read", actuals) ->
            let actuals = List.fold_left
                (fun actuals actual ->
                  let v = string_of_expr globals locals fdecl.fname
actual in
                      v :: actuals) [] actuals
                in
                if (List.length actuals) <> 0 then
                    raise (Failure ("function 'read' does not take
any parameter"
                        ^ ", called from function: '" ^ fname ^ "'"))
                else
                    { varname = "read" ; vartype = "string" }
      | Call(f, actuals) ->
            let fdecl =
                if f = "main" then
                    raise (Failure ("main function cannot be called
by " ^
                                               "function: '" ^ fname
^ "'"))
                else
                  try NameMap.find f func_decls
                      with Not_found -> raise (Failure ("Undefined
function: '"
                          ^ f ^ "' in function call: '" ^ fname ^
"'"))
                  in
                  if f <> fname then
                  let actuals = List.fold_left
                      (fun actuals actual ->
                        let v = string_of_expr globals locals
fdecl.fname actual in
                            v :: actuals) [] actuals
                      in
                            string_of_fdecl globals fdecl (List.rev
actuals);
                            { varname = fdecl.retname ; vartype =
fdecl.rettype }
                else
                    let actuals = List.fold_left
                      (fun actuals actual ->
                        let v = string_of_expr globals locals
fdecl.fname actual in
                            v :: actuals) [] actuals
                      in
                (try List.iter2 (fun formal actual ->
                    if formal.vartype <> actual.vartype then
                        raise (Failure ("Type mismatch in argument
passing between: '"
                        ^ formal.varname ^ "', type: " ^
formal.vartype ^ " and '"
                        ^ actual.varname ^ "', type: " ^
actual.vartype
                        ^ " in function: '" ^ fdecl.fname ^ "'")))
                       fdecl.formals (List.rev actuals)
                (* Check invalid number of arguments *)
                with Invalid_argument(_) ->
```

```
                    raise (Failure ("Wrong number of arguments passed
to function: '"
                        ^ fdecl.fname ^ "' from function: '" ^ fname
^ "'")));
                { varname = fdecl.retname ; vartype = fdecl.rettype }
    in

    let check_for_expr v l globals locals fname =
        if v.vartype <> "int" or v.vartype <> "float" or v.vartype <>
"double" then
            let varList = List.map (string_of_expr globals locals
fname) l
              in
          List.iter (fun var -> if var.vartype <> v.vartype then
                  raise (Failure ("For loop type mismatch between '"
^ v.varname
                  ^ "', type: " ^ v.vartype ^ " and '" ^ var.varname
                  ^ "', type: " ^ var.vartype ^ " in function: '" ^
fname ^ "'"))) varList;
          else
              raise (Failure ("Only int/float is allowed in the for
loop. Id '"
                  ^ v.varname ^ "', type: " ^ v.vartype ^ " found in
function: '" ^ fname ^ "'"))
    in

    let rec string_of_for_expr globals locals fname = function
        Assigna(v, e1, e2) ->
        if NameMap.mem v locals then
            let var = NameMap.find v locals in
            check_for_expr var [e1;e2;] globals locals fname
        else if NameMap.mem v globals then
              let var = NameMap.find v globals in
              check_for_expr var [e1;e2;] globals locals fname
          else raise (Failure ("Undeclared identifier in for loop: '"
^ v
              ^ "' in function: '" ^ fname ^ "'"))
    |   Assignb(v, e1, e2, e3) ->
        if NameMap.mem v locals then
            let var = NameMap.find v locals in
            check_for_expr var [e1;e2;e3;] globals locals fname
          else if NameMap.mem v globals then
              let var = NameMap.find v globals in
              check_for_expr var [e1;e2;e3] globals locals fname
        else raise (Failure ("Undeclared identifier in for loop: '" ^
v
            ^ "' in function: '" ^ fname ^ "'"))
    in
    let checkDimension kernel fname l =
        let rec loop l =
            if (List.length l) = 1 then
                List.length (List.hd l)
            else
                let c1 = List.length (List.hd l)
                and c2 = loop (List.tl l) in
                if (c1 <> c2) then
                    raise (Failure ("Invalid kernel initialization:
number of columns"
                    ^ " in all rows must be same. kernel: '" ^ kernel
^ "' ;"
                    ^"function: '" ^ fname ^ "'"))
                else c1
            in loop l
    in
    let checkKernelValues globals locals kernel fname l =
        let flatL = List.concat l in
            List.iter
            (fun exp ->
            let v = string_of_expr globals locals fname exp in
                if v.vartype <> "float" then
                    raise (Failure ("Invalid kernel value: all values
passed to "
                        ^ " kernel must be float type. kernel: '" ^
kernel ^ "' ;"
                    ^"function: '" ^ fname ^ "'"))) flatL
    in
    let blackhole e = ()
    in
    let rec string_of_stmt globals locals fname = function
        Block(stmts) ->
            List.iter (string_of_stmt globals locals fname) stmts
      | Expr(expr) -> let e = string_of_expr globals locals fname
expr in blackhole e
      | If(e, s, Block([])) -> let v = string_of_expr globals locals
fname e in
            if v.vartype <> "boolean" then
                raise (Failure ("if expression must be evaluated to
boolean, "
                    ^ "but here found with '" ^ v.vartype ^ "' ;"
                    ^ "function: '" ^ fname ^ "'"))
            else
                string_of_stmt globals locals fname s;
      | If(e, s1, s2) ->  let v = string_of_expr globals locals fname
e in
            if v.vartype <> "boolean" then
                raise (Failure ("if expression must be evaluated to
boolean, "
                    ^ "but here found with '" ^ v.vartype ^ "' ;"
                    ^ "function: '" ^ fname ^ "'"))
            else
                string_of_stmt globals locals fname s1;
                string_of_stmt globals locals fname s2;
      | Ifelseif(e, s, eif_list, Block([])) ->
            let v = string_of_expr globals locals fname e in
            if v.vartype <> "boolean" then
                raise (Failure ("if expression must be evaluated to
boolean, "
                    ^ "but here found with '" ^ v.vartype ^ "' ;"
                    ^ "function: '" ^ fname ^ "'"))
            else
                let rec string_of_elseif eif =
                      let en = string_of_expr globals locals fname
eif.elseif_expr in
                        if en.vartype <> "boolean" then
                            raise (Failure ("if expression must be
evaluated to boolean, "
                                ^ "but here found with '" ^ v.vartype
^ "' ;"
                                ^ "function: '" ^ fname ^ "'"))
                        else
                            string_of_stmt globals locals fname
eif.elseif_stmt;
```

```
                in
                    string_of_stmt globals locals fname s;
                    List.iter string_of_elseif eif_list;
      | Ifelseif(e, s1, eif_list, s2) ->
            let v = string_of_expr globals locals fname e in
            if v.vartype <> "boolean" then
                raise (Failure ("if expression must be evaluated to
boolean, "
                    ^ "but here found with '" ^ v.vartype ^ "' ;"
                    ^ "function: '" ^ fname ^ "'"))
            else
                let rec string_of_elseif eif =
                    let en = string_of_expr globals locals fname
eif.elseif_expr in
                        if en.vartype <> "boolean" then
                            raise (Failure ("if expression must be
evaluated to boolean, "
                                ^ "but here found with '" ^ v.vartype
^ "' ;"
                                ^ "function: '" ^ fname ^ "'"))
                        else
                            string_of_stmt globals locals fname
eif.elseif_stmt;
                in
                    string_of_stmt globals locals fname s1;
                    List.iter string_of_elseif eif_list;
                    string_of_stmt globals locals fname s2;
      | For(e, s) ->
                string_of_for_expr globals locals fname e;
                string_of_stmt globals locals fname s
      | While(e, s) -> let v = string_of_expr globals locals fname e
            in
            if v.vartype <> "boolean" then
                raise (Failure ("while expression must be evaluated
to boolean, "
                    ^ "but here found with '" ^ v.vartype ^ "' ;"
                    ^ "function: '" ^ fname ^ "'"))
            else
                string_of_stmt globals locals fname s;
      | KernelInit(id, valuesList) ->
            if NameMap.mem id locals then
                let varId = NameMap.find id locals in
                if varId.vartype <> "kernel" then
                  raise (Failure ("Expecting kernel type but got '" ^
varId.vartype ^
                      "' for variable: '" ^ varId.varname ^ " in
function: '" ^ fname ^ "'"))
              else
                  let col = checkDimension id fname valuesList in
                      blackhole col;
                      checkKernelValues globals locals id fname
valuesList
          else if NameMap.mem id globals then
                let varId = NameMap.find id globals in
                if varId.vartype <> "kernel" then
                  raise (Failure ("Expecting kernel type but got '" ^
varId.vartype ^
                      "' for variable: '" ^ varId.varname ^ " in
function: '" ^ fname ^ "'"))
              else
                  let col = checkDimension id fname valuesList in
```

```
                      blackhole col;
                      checkKernelValues globals locals id fname 
valuesList
          else raise (Failure ("Undeclared identifier found in 
function: '" ^ fname ^ "'"))

    in
    let locals =
        try List.fold_left2
            (fun locals formal actual ->
                if formal.vartype = actual.vartype then
                    if NameMap.mem formal.varname locals then
                        raise (Failure ("Formal variable: '" ^ 
formal.varname ^
                            "' has already been defined in the 
function: '" ^
                            fdecl.fname ^ "' parameter"))
                    else NameMap.add formal.varname formal locals
                else raise (Failure ("Type mismatch in argument 
passing between: '"
                      ^ formal.varname ^ "', type: " ^ formal.vartype 
^ " and '"
                      ^ actual.varname ^ "', type: " ^ actual.vartype
                      ^ " in function: '" ^ fdecl.fname ^ "'")))
                  NameMap.empty fdecl.formals actuals
          (* Check invalid number of arguments *)
        with Invalid_argument(_) ->
              raise (Failure ("Wrong number of arguments passed to 
function: '"
                  ^ fdecl.fname ^ "'"))
        in
        let locals = List.fold_left
              (fun locals local ->
                if NameMap.mem local.varname locals then
                        raise (Failure ("Local variable: '" ^ 
local.varname ^
                            "' has already been defined in the 
function: '" ^
                            fdecl.fname ^ "'"))
                else NameMap.add local.varname local locals) locals 
fdecl.locals
            in
            if fdecl.retname <> "" then
                if NameMap.mem fdecl.retname locals then
                    raise (Failure ("Return variable: '" ^ 
fdecl.retname ^
                        "' has already been defined in the function: 
'" ^
                        fdecl.fname ^ "'"))
                else
                    let ret = { varname = fdecl.retname; vartype = 
fdecl.rettype } in
                    let locals = NameMap.add ret.varname ret locals 
in
                    List.iter (string_of_stmt globals locals 
fdecl.fname) fdecl.body
            else
                List.iter (string_of_stmt globals locals fdecl.fname) 
fdecl.body
    in
    let globals = List.fold_left
```

```
        (fun globals vdecl ->
            if NameMap.mem vdecl.varname globals then
                raise (Failure ("Global variable : '" ^ vdecl.varname
^
                            "' has already been defined."))
            else
                NameMap.add vdecl.varname vdecl globals)
NameMap.empty vars
    in
        try
            string_of_fdecl globals (NameMap.find "main" func_decls)
[]
        with Not_found ->
            raise (Failure ("There is no main() function"))
```

javaprinter.ml

```
open Ast

let import_decl = "
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.color.ColorSpace;
import java.awt.image.BufferedImage;
import java.awt.image.ColorConvertOp;
import java.awt.image.ColorModel;
import java.awt.image.ConvolveOp;
import java.awt.image.Kernel;
import java.awt.image.WritableRaster;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.RuntimeException;
import java.util.Hashtable;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

"

let class_decl = "public class Matlip {\n" ^
                 "enum OPERATION { ADD, SUB, MUL, DIV }" ^
                 "\n" ^
                 "   static BufferedImage imread(String path){\n" ^
                 "      try{\n" ^
                 "         BufferedImage  image;\n" ^
                 "         image = ImageIO.read(new File(path));\n" ^
                 "         return image;\n" ^
                 "      }\n" ^
                 "      catch (IOException e){\n" ^
                 "
System.out.println(\"Error:\"+e.getMessage());\n" ^
                 "         return null;\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^
                 "   static void imshow(BufferedImage im){\n" ^
                 "\n" ^
                 "      try {\n" ^
                 "         ImageRenderComponent irc = new
ImageRenderComponent(im);\n" ^
                 "         irc.setOpaque(true);\n" ^
                 "         JFrame f = new JFrame(\"Display
Image\");\n" ^
                 "
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);\n" ^
                 "         f.setContentPane(irc);\n" ^
                 "         f.setSize(800,600);\n" ^
                 "         f.setVisible(true);\n" ^
                 "      }\n" ^
                 "      catch (Exception e){\n" ^
                 "         e.printStackTrace();\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^
                 "   /*\n" ^
                 "    * the image formats supported by JRE are\n" ^
                 "    * BMP, bmp\n" ^
                 "      jpg, JPG\n" ^
                 "      wbm\n" ^
                 "      jpeg\n" ^
                 "      png, PNG\n" ^
                 "      JPEG\n" ^
                 "      WBMP\n" ^
                 "      GIF, gif\n" ^
                 "    */\n" ^
                 "   static void imsave(BufferedImage im,String
path){\n" ^
                 "      try {\n" ^
                 "         File outputfile = new File(path);\n" ^
                 "         String format =
path.substring(path.lastIndexOf(\".\")+1, path.length());\n" ^
                 "         ImageIO.write(im, format, outputfile);\n"
^
                 "      }\n" ^
                 "      catch (IOException e) {\n" ^
                 "
System.out.println(\"Error:\"+e.getMessage());\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^ "
    static BufferedImage imnew(int width, int height, String type){
        BufferedImage im=null;

        if (type.contentEquals(\"RGB\")){
            im = new BufferedImage(width, height,
BufferedImage.TYPE_3BYTE_BGR);
            for (int i=0;i<im.getHeight();i++){
                for (int j=0;j<im.getWidth();j++){
                    im.setRGB(j, i, 0xFF000000);
                }
            }

        }
        else if (type.contentEquals(\"GREY\") ||
type.contentEquals(\"GRAY\")){
            im = new BufferedImage(width, height,
BufferedImage.TYPE_BYTE_GRAY);
        }
        return im;
    }
                 " ^
                 "\n" ^
                 "
    static Kernel kernelinit(){
        Kernel k = new Kernel(3,3,new float[] {
                    0.0f, 0.0f, 0.0f,
                    0.0f, 1.0f, 0.0f,
                    0.0f, 0.0f, 0.0f
                });
        return k;
    }

    static Kernel kernelnew(int width, int height){

        try{
            float[] data = new float[width*height];
            Kernel k = new Kernel(width, height, data);
            return k;
        }
        catch (Exception e){
            e.printStackTrace();
            return null;
        }
    }
                 " ^
                 "\n" ^
                 "   static void print(Object data){\n" ^
                 "      System.out.print(data);\n" ^
                 "   }\n" ^
                 "\n" ^
                 "   static int getWidth(BufferedImage im){\n" ^
                 "      try {\n" ^
                 "         return im.getWidth();\n" ^
                 "      }\n" ^
                 "      catch (Exception e){\n" ^
                 "         e.printStackTrace();\n" ^
                 "         return -1;\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^
                 "   static int getWidth(Kernel k){\n" ^
                 "      try {\n" ^
                 "         return k.getWidth();\n" ^
                 "      }\n" ^
                 "      catch (Exception e){\n" ^
                 "         e.printStackTrace();\n" ^
                 "         return -1;\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^
                 "   static int getHeight(BufferedImage im){\n" ^
                 "      try {\n" ^
                 "         return im.getHeight();\n" ^
                 "      }\n" ^
                 "      catch (Exception e){\n" ^
                 "         e.printStackTrace();\n" ^
                 "         return -1;\n" ^
                 "      }\n" ^
```

```
                 "   }\n" ^
                 "\n" ^
                 "   static int getHeight(Kernel k){\n" ^
                 "      try {\n" ^
                 "         return k.getHeight();\n" ^
                 "      }\n" ^
                 "      catch (Exception e){\n" ^
                 "         e.printStackTrace();\n" ^
                 "         return -1;\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^ "
    static String getType(BufferedImage im){

        try{
            if (im.getType() == BufferedImage.TYPE_INT_ARGB ||
im.getType() == BufferedImage.TYPE_3BYTE_BGR || im.getType() ==
BufferedImage.TYPE_BYTE_INDEXED){
                return \"RGB\";
            }
            else if (im.getType() == BufferedImage.TYPE_BYTE_GRAY) {
                return \"GREY\";
            }
            else {
                return \"unknown type\";
            }
        }
        catch (Exception e){
            e.printStackTrace();
            return \"ERROR: getType()\";
        }
    }
                 " ^
                 "\n" ^
                 "   static int getLength(String s){\n" ^
                 "      try {\n" ^
                 "         return s.length();\n" ^
                 "      }\n" ^
                 "      catch (Exception e){\n" ^
                 "         e.printStackTrace();\n" ^
                 "         return -1;\n" ^
                 "      }\n" ^
                 "   }\n" ^
                 "\n" ^
                 "
    static float getKernelData(Kernel k,int i, int j){
        float[] data = new float[k.getHeight()*k.getWidth()];
        k.getKernelData(data);

        return data[i+j*k.getWidth()];

    }

    static Kernel setKernelData(Kernel k, int i, int j, float value){
        float[] data = new float[k.getHeight()*k.getWidth()];
        k.getKernelData(data);
        data[i+j*k.getWidth()] = value;
        k = new Kernel(k.getWidth(),k.getHeight(),data);
        return k;
    }
```

```
   private static int objectToInt(Object o){
       if
(o.getClass().getName().contentEquals(\"java.lang.Float\")){
           return (int)Float.parseFloat(o.toString());
       }
       else if
(o.getClass().getName().contentEquals(\"java.lang.Double\")){
           return (int)Double.parseDouble(o.toString());
       }
       else if
(o.getClass().getName().contentEquals(\"java.lang.Integer\")){
           return Integer.parseInt(o.toString());
       }
       else if
(o.getClass().getName().contentEquals(\"java.lang.String\")){
           return Integer.parseInt(o.toString());
       }
       else {
           throw new RuntimeException(\"type error!\");
       }

   }

   private static float objectToFloat(Object o){
       if
(o.getClass().getName().contentEquals(\"java.lang.Float\")){
           return Float.parseFloat(o.toString());
       }
       else if
(o.getClass().getName().contentEquals(\"java.lang.Double\")){
           return (float)Double.parseDouble(o.toString());
       }
       else if
(o.getClass().getName().contentEquals(\"java.lang.Integer\")){
           return (float)Integer.parseInt(o.toString());
       }
       else if
(o.getClass().getName().contentEquals(\"java.lang.String\")){
           return Float.parseFloat(o.toString());
       }
       else {
           throw new RuntimeException(\"type error!\");
       }
   }

    static Object doArithmetic(Object op1, Object op2, OPERATION op){

            if (op1 instanceof BufferedImage){
                BufferedImage des;
                boolean color=false;
                try{
                    int t =
((BufferedImage)op1).getSampleModel().getSample(0, 0, 1,
((BufferedImage)op1).getRaster().getDataBuffer());
                    //des =
imnew(((BufferedImage)op1).getWidth(),((BufferedImage)op1).getHeight(
),\"RGB\");
                    des = new
BufferedImage(((BufferedImage)op1).getWidth(),((BufferedImage)op1).ge
tHeight(),((BufferedImage)op1).getType());
                    color=true;
                } catch (Exception e){
                    des = 
imnew(((BufferedImage)op1).getWidth(),((BufferedImage)op1).getHeight(
),\"GREY\");
                }

                if (op2 instanceof BufferedImage){
                    if 
(((BufferedImage)op1).getHeight()!=((BufferedImage)op2).getHeight()
|| ((BufferedImage)op1).getWidth()!=((BufferedImage)op2).getWidth())
                        throw new RuntimeException(\"dimension 
mismatch!\");
                }



                int v1;
                int v2;
                for (int j=0;j<((BufferedImage)des).getHeight();j++){
                    for (int 
i=0;i<((BufferedImage)des).getWidth();i++){
                        if (color){
                            v1 = ((BufferedImage)op1).getRGB(i, j);
                            if (op2 instanceof BufferedImage)
                                v2 = ((BufferedImage)op2).getRGB(i, 
j);
                            else
                                v2 = objectToInt(op2);


                        }
                        else{
                            v1 = 
((BufferedImage)op1).getSampleModel().getSample(i,j,0,((BufferedImage
)op1).getRaster().getDataBuffer());
                            if (op2 instanceof BufferedImage)
                                v2 = 
((BufferedImage)op2).getSampleModel().getSample(i,j,0,((BufferedImage
)op2).getRaster().getDataBuffer());
                            else
                                v2 = objectToInt(op2);
                        }

                        switch (op){
                            case ADD:
                                if (color){
                                    ((BufferedImage)des).setRGB(i, j, 
(int)v1+v2);
                                    break;
                                }
                                else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1+v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                    break;
                                }
                            case SUB:
                               if (color){
                                    ((BufferedImage)des).setRGB(i, j, 
(int)v1-v2);
                                    break;
                                }
                                else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1-v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                    break;
                                }
                            case MUL:
                                if (color){
                                    ((BufferedImage)des).setRGB(i, j,
(int)v1*v2);
                                    break;
                                }
                                else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1*v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                    break;
                                }
                            case DIV:
                               if (color){
                                    ((BufferedImage)des).setRGB(i, j,
(int)v1/v2);
                                    break;
                                }
                                else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1/v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                    break;
                                }
                        }
//                        if
(!((BufferedImage)des).isAlphaPremultiplied())
//
((BufferedImage)des).getSampleModel().setSample(i, j, 3,0xFF ,
((BufferedImage)des).getRaster().getDataBuffer());
                    }
                } // end of for
                return des;
            }

            else if (op1 instanceof Kernel){
                float[] des = new
float[((Kernel)op1).getWidth()*((Kernel)op1).getHeight()];
                float[] data1 = new
float[((Kernel)op1).getWidth()*((Kernel)op1).getHeight()];
                ((Kernel)op1).getKernelData(data1);

                float[] data2 = new
float[((Kernel)op1).getWidth()*((Kernel)op1).getHeight()];

                if (op2 instanceof Kernel){
                    if
(((Kernel)op1).getHeight()!=((Kernel)op2).getHeight() ||
((Kernel)op1).getWidth()!=((Kernel)op2).getWidth())
                        throw new RuntimeException(\"dimension
mismatch!\");


                    ((Kernel)op2).getKernelData(data2);
                }


                float v1;
                float v2;

                for (int i=0;i<des.length;i++){
                    v1 = data1[i];
                    if (op2 instanceof Kernel)
                        v2 = data2[i];
                    else
                        v2 = objectToFloat(op2);

                      switch (op){
                            case ADD:
                               des[i] = v1+v2;
                               break;
                            case SUB:
                                des[i] = v1-v2;
                                break;
                            case MUL:
                                  des[i] = v1*v2;
                                  break;
                            case DIV:
                              des[i] = v1/v2;
                               break;
                        }
                }

                return new
Kernel(((Kernel)op1).getWidth(),((Kernel)op1).getHeight(),des);
            }
            else{
                throw new RuntimeException(\"op1 type error!\");
            }
    }


    static Object clone(Object src){
        if (src instanceof BufferedImage){
             String[] pnames =
((BufferedImage)src).getPropertyNames();
             Hashtable<String, Object> cproperties = new
Hashtable<String,Object>();
             if(pnames != null) {
               for(int i = 0; i < pnames.length; i++) {
                   cproperties.put(pnames[i],
((BufferedImage)src).getProperty(pnames[i]));
               }
            }
            WritableRaster wr = ((BufferedImage)src).getRaster();
            WritableRaster cwr = wr.createCompatibleWritableRaster();
            cwr.setRect(wr);
            BufferedImage des = new BufferedImage(
               ((BufferedImage)src).getColorModel(),  // should be
immutable
               cwr,
               ((BufferedImage)src).isAlphaPremultiplied(),
               cproperties);

            return des;
        }
        else if (src instanceof Kernel){
            return ((Kernel)src).clone();
        }
        else{
            throw new RuntimeException(\"the type of source is not
cloneable!\");
        }
    }

    static String read(){
        try{
            return (new BufferedReader(new
InputStreamReader(System.in))).readLine();
        }
        catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }

    static void setImagePixel(BufferedImage im, int col, int row,
String channel, int value){
        try{
            if (channel.toUpperCase().contentEquals(\"RGB\")){
                    im.setRGB(col, row, value|0xFF000000);
            }
            else if (channel.toUpperCase().contentEquals(\"R\")){
                    im.getSampleModel().setSample(col, row, 0, value,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals(\"G\")){
                    im.getSampleModel().setSample(col, row, 1, value,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals(\"B\")){
                    im.getSampleModel().setSample(col, row, 2, value,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals(\"GREY\") ||
channel.toUpperCase().contentEquals(\"GRAY\")){
                    im.getSampleModel().setSample(col, row, 0, value,
im.getRaster().getDataBuffer());
            }
            else{
                throw new RuntimeException(\"invalid channnel!\");
            }
        }
        catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        }
    }

    static int getImagePixel(BufferedImage im, int col, int row,
String channel){
        int ret;
        try{
            if (channel.toUpperCase().contentEquals(\"RGB\")){
                ret = im.getRGB(col, row)&0x00FFFFFF;
            }
            else if (channel.toUpperCase().contentEquals(\"R\")){
                    ret = im.getSampleModel().getSample(col, row, 0,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals(\"G\")){
                    ret = im.getSampleModel().getSample(col, row, 1,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals(\"B\")){
                    ret = im.getSampleModel().getSample(col, row, 2,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals(\"GREY\") ||
channel.toUpperCase().contentEquals(\"GRAY\")){
                    ret = im.getSampleModel().getSample(col, row, 0,
im.getRaster().getDataBuffer());
            }
            else{
                throw new RuntimeException(\"invalid channnel!\");
            }
        }
        catch(Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        finally {
            return ret;
        }
    }

    static BufferedImage convolve(BufferedImage im, Kernel k){
        BufferedImage imc = new BufferedImage(im.getWidth(),
im.getHeight(), im.getType());
        ConvolveOp convo = new ConvolveOp(k);
        convo.filter(im, imc);
        return imc;
    }
                 "

(* initialize all variables *)
let initial_value vtype =
  if vtype = "void" then ""
  else if vtype = "int" then "0"
  else if vtype = "float" then "(float)0.0"
  else if vtype = "string" then "\"\""
  else if vtype = "boolean" then "false"
  else if vtype = "image" then "imnew(100, 100, \"RGB\")"
  else if vtype = "kernel" then "kernelinit()"
  else
    raise (Failure ("unrecognized type: " ^ vtype))

(* convert MATLIP data type to Java data type *)
let java_type_of vtype =
  if vtype = "void" then "void"
  else if vtype = "int" then "int"
  else if vtype = "float" then "float"
  else if vtype = "string" then "String"
  else if vtype = "boolean" then "boolean"
  else if vtype = "image" then "BufferedImage"
  else if vtype = "kernel" then "Kernel"
  else
```

```
    raise (Failure ("unrecognized type: " ^ vtype))

let fsignature fdecl =
  if fdecl.fname = "main" then "public static void main(String[]
args)"
  (* else "public static int " ^ fdecl.fname ^ "(int " ^
String.concat ", int " fdecl.formals ^ ")" *)
  else
    let formal_string_list = List.fold_left (fun f_list formal ->
(java_type_of formal.vartype ^ " " ^ formal.varname)::f_list) []
fdecl.formals
    in
    "public static " ^ java_type_of fdecl.rettype ^ " " ^ fdecl.fname
^ "(" ^ String.concat "," (List.rev formal_string_list) ^ ")"

let imgrender_decl = "class ImageRenderComponent extends JPanel {\n"
^
                     "   BufferedImage image;\n" ^
                     "   Dimension size;\n" ^
                     "\n" ^
                     "   public ImageRenderComponent(BufferedImage
image) {\n" ^
                     "      this.image = image;\n" ^
                     "      size = new Dimension(image.getWidth(),
image.getHeight());\n" ^
                     "   }\n" ^
                     "\n" ^
                     "   @Override\n" ^
                     "   protected void paintComponent(Graphics g)
{\n" ^
                     "      super.paintComponent(g);\n" ^
                     "      int x = (getWidth() - size.width)/2;\n" ^
                     "      int y = (getHeight() - size.height)/2;\n"
^
                     "      g.drawImage(image, x, y, this);\n" ^
                     "   }\n" ^
                     "\n" ^
                     "   @Override\n" ^
                     "   public Dimension getPreferredSize() {\n" ^
                     "      return size;\n" ^
                     "   }\n" ^
                     "}\n"

let fcall f =
  if f = "print" then "System.out.println"
  else if f = "imread" then "imread"
  else if f = "imsave" then "imsave"
  else if f = "imshow" then "imshow"
  else if f = "kernelnew" then "kernelnew"
  else if f = "getwidth" then "getWidth"
  else if f = "getheight" then "getHeight"
  else if f = "getlength" then "getLength"
  else if f = "gettype" then "getType"
  else if f = "toint" then "objectToInt"
  else if f = "tofloat" then "objectToFloat"
  else if f = "read" then "read"
  else f

(* build a list of 'num' elements of name *)
let rec build_list (name, num, l) =
  if num = 0 then l
  else build_list (name, num-1, name::l)

(* determine channel number *)
let channel_num s =
  if s = "R" || s = "r" then "0"
  else if s = "G" || s = "g" then "1"
  else if s = "B" || s = "b" then "2"
  else raise (Failure ("Unrecognized channel: must be one of \"R\",
\"G\", \"B\", \"RGB\""))

(* returns the type of an expression, mainly for images, kernels, and
power operator *)
let rec type_of_expr symbols = function
    Literal(l) -> { varname = string_of_int l; vartype = "int" }
  | Floatlit(f) -> { varname = string_of_float f; vartype = "float" }
  | String(s) -> { varname = s; vartype = "string" }
  | Bool(s) -> { varname = s; vartype = "boolean" }
  | Id(s) ->
      (* try to find it in local table first *)
      let exists = List.exists (fun a -> if a.varname = s then true
else false) symbols
      in
      if exists then
        let var = List.find (fun a -> if a.varname = s then true else
false) symbols
        in
        var
      else (* then try to find it in global table *)
        let exists2 = List.exists (fun a -> if a.varname = s ^
"_global_var" then true else false) symbols
        in
        if exists2 then
          let var = List.find (fun a -> if a.varname = s ^
"_global_var" then true else false) symbols
          in
          { varname = s; vartype = var.vartype }
        else
          raise (Failure ("Unable to find " ^ s ^ " in symbol table"))
      (**
      let var_list = List.find_all (fun a -> if a.varname = s then
true else false) symbols
      in
      if List.length var_list > 1 then
        raise (Failure ("Functions and variables cannot share the
same name: '" ^ s ^ "'"))
      else
        List.hd var_list
      **)
  | Uminus(e) ->
      let ue = type_of_expr symbols e
      in ue
  | Not(e) ->
      let ne = type_of_expr symbols e
      in ne
  | Binop(e1, o, e2) -> begin
      match o with
        Equal | Neq | Less | Leq | Greater | Geq ->
        { varname = "null"; vartype = "boolean" }
      | _ -> (* The rest of expr have the same type as the first
operand *)
        let e = type_of_expr symbols e1
        in e
    end
  | Imnew(row, col, format) -> { varname = "imnew"; vartype =
"image" }
  | Kernelnew(row, col) -> { varname = "kernelnew";vartype =
"kernel" }
  | Assign(v, e) ->
      (* try to find it in local table first *)
      let exists = List.exists (fun a -> if a.varname = v then true
else false) symbols
      in
      if exists then
        let var = List.find (fun a -> if a.varname = v then true else
false) symbols
        in
        var
      else (* then try to find it in global table *)
        let exists2 = List.exists (fun a -> if a.varname = v ^
"_global_var" then true else false) symbols
        in
        if exists2 then
          let var = List.find (fun a -> if a.varname = v ^
"_global_var" then true else false) symbols
          in
          { varname = v; vartype = var.vartype }
        else
          raise (Failure ("Unable to find " ^ v ^ " in symbol table"))
      (**
      let var_list = List.find_all (fun a -> if a.varname = v then
true else false) symbols
      in
      if List.length var_list > 1 then
        raise (Failure ("Functions and variables cannot share the
same name: '" ^ v ^ "'"))
      else
        List.hd var_list
      **)
  | AssignPixel(id, row, col, channel, value) -> { varname = "null";
vartype = "int" }
  | AssignKernel(id, row, col, value) -> { varname = "null";vartype =
"float" }
  | ImageAccess(id, row, col, channel) -> { varname = "null"; vartype
= "int" }
  | KernelAccess(id, row, col) -> { varname = "null";vartype =
"float" }
  | Call(f, el) ->
      let exists = List.exists (fun a -> if a.varname = f ^
"_global_func" then true else false) symbols
      in
      if exists then (* if f is in our symbole table *)
        let ff = List.find (fun a -> if a.varname = f ^
"_global_func" then true else false) symbols
        in
        { varname = f; vartype = ff.vartype }
        (**
        let ff_list = List.find_all (fun a -> if a.varname = f then
true else false) symbols
        in
        if List.length ff_list > 1 then
          raise (Failure ("Functions and variables cannot share the
same name: '" ^ f ^ "'"))
```

```
        else
          let ff = List.hd ff_list in
          { varname = ff.varname; vartype = ff.vartype }
        **)
      else if f = "imread" then
        { varname = "imread"; vartype = "image" }
      else if f = "imnew" then
        { varname = "imnew"; vartype = "image" }
      else if f = "kernelnew" then
        { varname = "kernelnew"; vartype = "kernel" }
      else if f = "getwidth" then
        { varname = "getwidth"; vartype = "int" }
      else if f = "getheight" then
        { varname = "getheight"; vartype = "int" }
      else if f = "getlength" then
        { varname = "getlength"; vartype = "int" }
      else if f = "gettype" then
        { varname = "gettype"; vartype = "string" }
      else if f = "toint" then
        { varname = "toint"; vartype = "int" }
      else if f = "tofloat" then
        { varname = "tofloat"; vartype = "float" }
      else if f = "tostring" then
        { varname = "tostring"; vartype = "string" }
      else if f = "togray" then
        { varname = "togray"; vartype = "image" }
      else if f = "read" then
        { varname = "read"; vartype = "string" }
      else if f = "mod" then
        let first_arg = List.hd el
        in
        let e = type_of_expr symbols first_arg
        in
        if e.vartype = "int" then
          { varname = "mod"; vartype = "int" }
        else if e.vartype = "float" then
          { varname = "mod"; vartype = "float" }
        else (* walker should block this*)
          { varname = ""; vartype = "" }
      else if f = "sqrt" then
        let arg = List.hd el
        in
        let e = type_of_expr symbols arg
        in
        if e.vartype = "int" then
          { varname = "sqrt"; vartype = "int" }
        else if e.vartype = "float" then
          { varname = "sqrt"; vartype = "float" }
        else (* walker should block this*)
          { varname = ""; vartype = "" }
      else
        { varname = ""; vartype = "" }
  | _ -> { varname = ""; vartype = "" }

(* given variable v, find it in symbol table *)
let rec find_symbol symbols v =
  (* try to find it in local table first *)
  let exists = List.exists (fun a -> if a.varname = v then true else
false) symbols
  in
  if exists then
```

```
    let var = List.find (fun a -> if a.varname = v then true else
false) symbols
    in
    var
  else (* then try to find it in global table *)
    let exists2 = List.exists (fun a -> if a.varname = v ^
"_global_var" then true else false) symbols
    in
    if exists2 then
      let var = List.find (fun a -> if a.varname = v ^ "_global_var"
then true else false) symbols
      in
      { varname = v; vartype = var.vartype }
    else
      raise (Failure ("Unable to find " ^ v ^ " in symbol table"))

let rec string_of_expr symbols = function
    Literal(l) -> string_of_int l
  | Floatlit(l) -> "(float)" ^ string_of_float l (* cast to
float..Java's default type is double *)
  | String(s) -> "\"" ^ s ^ "\""
  | Bool(s) -> s
  | Id(s) -> s
  | Uminus(e) -> "-(" ^ string_of_expr symbols e ^ ")"
  | Not(e) -> "!(" ^ string_of_expr symbols e ^ ")"
  | Binop(e1, o, e2) -> (* need to handle special cases here where e1
is an image or kernel type *)
      let e1_decl = type_of_expr symbols e1
      in
      let e2_decl = type_of_expr symbols e2
      in
      if e1_decl.vartype = "image" then (* image type can do +, -, *,
/, @ *)
        let op =
          (match o with
             Add -> "OPERATION.ADD" | Sub -> "OPERATION.SUB" |
             Mult -> "OPERATION.MUL" | Div -> "OPERATION.DIV" |
             Convolve -> "Convolve" |
             Equal | Neq | Less | Leq | Greater | Geq |
             And | Or | Power ->
               raise (Failure ("The only arithmetic operations
allowed for images are +, -, *, /, @")))
        in
        if op = "Convolve" then
          if e2_decl.vartype = "kernel" then
            (***
            "new ConvolveOp(" ^ string_of_expr symbols e2 ^
                                "",
ConvolveOp.EDGE_NO_OP,null).filter(" ^
                                string_of_expr symbols e1 ^ ", null)"
            ***)
            "convolve(" ^ string_of_expr symbols e1 ^ ", " ^
                        string_of_expr symbols e2 ^ ")"
          else
            raise (Failure ("The second operand of '@' must be a
kernel type"))
        else
        "(BufferedImage)doArithmetic(" ^ string_of_expr symbols e1 ^
", " ^
                                   string_of_expr symbols e2 ^
", " ^
```

```
                                         op ^ ")"
      else if e1_decl.vartype = "kernel" then
        let op =
          (match o with
             Add -> "OPERATION.ADD" | Sub -> "OPERATION.SUB" |
             Mult -> "OPERATION.MUL" | Div -> "OPERATION.DIV" |
             Equal | Neq | Less | Leq | Greater | Geq |
             Convolve | And | Or | Power ->
               raise (Failure ("The only arithmetic operations
allowed for kernels are +, -, *, /")))
        in
        "(Kernel)doArithmetic(" ^ string_of_expr symbols e1 ^ ", " ^
                                  string_of_expr symbols e2 ^ ", " ^
                                  op ^ ")"
      else
        let op =
          (match o with
             Power -> "POWER"
           | _ -> "")
        in
        if op = "POWER" then
          if e1_decl.vartype = "int" && e2_decl.vartype = "int" then
            "(int)Math.pow(" ^ string_of_expr symbols e1 ^ ", " ^
                               string_of_expr symbols e2 ^ ")"
          else if e1_decl.vartype = "float" && e2_decl.vartype =
"float" then
            "(float)Math.pow(" ^ string_of_expr symbols e1 ^ ", " ^
                                 string_of_expr symbols e2 ^ ")"
          else "" (* walker should block this *)
        else
          string_of_expr symbols e1 ^ " " ^
          (match o with
           Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
           | Equal -> "==" | Neq -> "!="
           | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
           | And -> "&&" | Or -> "||"
           | Power -> "" (* this should never match! *)
           | Convolve -> raise (Failure ("'@' operator can only be
used with images and kernels"))) ^ " " ^
           string_of_expr symbols e2
  (*****
  | Assign(v, e) -> begin
      match e with
        Id(s) ->
          let lvar = List.find (fun a -> if a.varname = v then true
else false) symbols
          in
          let rvar = List.find (fun a -> if a.varname = s then true
else false) symbols
          in
          if lvar.vartype = "image" && rvar.vartype = "image" then
            (* clone an image *)
            v ^ " = (BufferedImage)clone(" ^ s ^ ")"
          else if lvar.vartype = "kernel" && rvar.vartype = "kernel"
then
            (* clone a kernel *)
            v ^ " = (Kernel)clone(" ^ s ^ ")"
          else
            v ^ " = " ^ string_of_expr symbols e
      | _ -> v ^ " = " ^ string_of_expr symbols e
    end
  *****)
  | Assign(v, e) ->
      (*let lvalue = List.find (fun a -> if a.varname = v then true
else false) symbols in*)
      let lvalue = find_symbol symbols v in
      let rvalue = type_of_expr symbols e in
      if lvalue.vartype = "image" && rvalue.vartype = "image" then
        if rvalue.varname = "imnew" || rvalue.varname = "imread" ||
rvalue.varname = "togray" then
          v ^ " = " ^ string_of_expr symbols e (* no need to clone if
it's a function call *)
        else
        (* clone an image *)
        v ^ " = (BufferedImage)clone(" ^ string_of_expr symbols e ^
")"
      else if lvalue.vartype = "kernel" && rvalue.vartype = "kernel"
then
        if rvalue.varname = "kernelnew" then
          v ^ " = " ^ string_of_expr symbols e (* no need to clone if
it's a function call *)
        else
        (* clone a kernel *)
        v ^ " = (Kernel)clone(" ^ string_of_expr symbols e ^ ")"
      else
        v ^ " = " ^ string_of_expr symbols e

  | Call(f, el) ->
      let symbols_list = build_list (symbols, List.length el, [])
      in
      (*let ff = List.find (fun a -> if a.varname = f then true else
false) symbols
      in
      ff.varname ^ ff.vartype ^ *)
      if f = "mod" then
        let e1 = List.nth el 0 in
        let e2 = List.nth el 1 in
        string_of_expr symbols e1 ^ " % " ^ string_of_expr symbols e2
      else if f = "tostring" then
        let e = List.nth el 0 in
          "String.valueOf(" ^ string_of_expr symbols e ^ ")"
      else if f = "togray" then
        let e = List.nth el 0 in
          "(new
ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY),null)).filt
er(" ^
          string_of_expr symbols e ^ ", null)"
      else if f = "sqrt" then
        let e = List.nth el 0 in
        let e_decl = type_of_expr symbols e in
        if e_decl.vartype = "int" then
          "(int)Math.sqrt(" ^ string_of_expr symbols e ^ ")"
        else if e_decl.vartype = "float" then
          "(float)Math.sqrt(" ^ string_of_expr symbols e ^ ")"
        else (* walker should block this *)
          ""
      else
        fcall f ^ "(" ^ String.concat ", " (List.map2 string_of_expr
symbols_list el) ^ ")"
  | Datatype(_) -> ""
  | Imnew(e1, e2, s) ->
      let channel =
        if s = "rgb" || s = "RGB" then "RGB"
        else if s = "grey" || s = "GREY" || s = "gray" || s = "GRAY"
then "GREY"
        else "RGB" (* default image type *)
      in
      "imnew(" ^ string_of_expr symbols e1 ^ ", " ^
                 string_of_expr symbols e2 ^ ", " ^
                 "\"" ^ channel ^ "\")"
  (*****
  | ImageAccess(var, e1, e2, s) ->
      if s = "rgb" || s = "RGB" then
        var ^ ".getRGB(" ^ string_of_expr symbols e1 ^ ", " ^
                           string_of_expr symbols e2 ^ ")"
      else if s = "gray" || s = "grey" || s = "GRAY" || s = "GREY"
then
        var ^ ".getSampleModel().getSample(" ^ string_of_expr symbols
e1 ^ ", " ^
                                               string_of_expr symbols
e2 ^ ", " ^
                                               "0" ^ ", " ^
                                               var ^
".getRaster().getDataBuffer())"
      else
        var ^ ".getSampleModel().getSample(" ^ string_of_expr symbols
e1 ^ ", " ^
                                               string_of_expr symbols
e2 ^ ", " ^
                                               channel_num s ^ ", " ^
                                               var ^
".getRaster().getDataBuffer())"
  *****)
  | ImageAccess(var, e1, e2, e3) ->
      "getImagePixel(" ^ var ^ ", " ^
                         string_of_expr symbols e1 ^ ", " ^
                         string_of_expr symbols e2 ^ ", " ^
                         string_of_expr symbols e3 ^ ")"
  | Kernelnew(e1, e2) ->
      "kernelnew(" ^ string_of_expr symbols e1 ^ ", " ^
                     string_of_expr symbols e2 ^ ")"
  | KernelAccess(k, e1, e2) ->
      "getKernelData(" ^ k ^ ", " ^
                         string_of_expr symbols e1 ^ ", " ^
                         string_of_expr symbols e2 ^ ")"
  (*****
  | Convolve(im, k) ->
      "new ConvolveOp(" ^ k ^ ", ConvolveOp.EDGE_NO_OP,null).filter("
^ im ^ ", null)"
  *****)
  (*****
  | AssignPixel(var, e1, e2, s, value) ->
      if s = "rgb" || s = "RGB" then
        var ^ ".setRGB(" ^ string_of_expr symbols e1 ^ ", " ^
                           string_of_expr symbols e2 ^ ", " ^
                           string_of_expr symbols value ^ ")"
      else if s = "gray" || s = "grey" || s = "GRAY" || s = "GREY"
then
        var ^ ".getSampleModel().setSample(" ^ string_of_expr symbols
e1 ^ ", " ^
                                               string_of_expr symbols
e2 ^ ", " ^
                                               "0" ^ ", " ^
```

```
                                                string_of_expr symbols
value ^ ", " ^
                                                var ^
".getRaster().getDataBuffer())"
      else
        var ^ ".getSampleModel().setSample(" ^ string_of_expr symbols
e1 ^ ", " ^
                                                string_of_expr symbols
e2 ^ ", " ^
                                                channel_num s ^ ", " ^
                                                string_of_expr symbols
value ^ ", " ^
                                                var ^
".getRaster().getDataBuffer())"
  *****)
  | AssignPixel(var, e1, e2, e3, value) ->
      "setImagePixel(" ^ var ^ ", " ^
                         string_of_expr symbols e1 ^ ", " ^
                         string_of_expr symbols e2 ^ ", " ^
                         string_of_expr symbols e3 ^ ", " ^
                         string_of_expr symbols value ^ ")"
  | AssignKernel(var, e1, e2, value) ->
      var ^ " = setKernelData(" ^ var ^ ", " ^
                                  string_of_expr symbols e1 ^ ", " ^
                                  string_of_expr symbols e2 ^ ", " ^
                                  string_of_expr symbols value ^ ")"

(* convert 2D matrix to 1D matrix for kernel *)
(* e.g. 3x3 to 1x9 *)
let rec expand_kernel_list a l =
  let lr = List.rev l
  in a @ lr

(*let rec string_of_stmt retname = function*)
let rec string_of_stmt symbols = function
    Block(stmts) ->
      (*let retname_list = build_list (retname, List.length stmts,
[])*)
      let symbols_list = build_list (symbols, List.length stmts, [])
      in
      (*"{\n" ^ String.concat "" (List.map2 string_of_stmt
retname_list stmts) ^ "}\n"*)
      "{\n" ^ String.concat "" (List.map2 string_of_stmt symbols_list
stmts) ^ "}\n"
  | Expr(expr) -> begin
      match expr with
        (* we ignore stmts that don't have lvalue to store the result
of an expr *)
        Literal(_) | Floatlit(_) | String(_) | Id(_) | Uminus(_) |
Binop(_, _, _) -> ""
      | _ -> string_of_expr symbols expr ^ ";\n";
    end
  | If(e, s, Block([])) -> "if (" ^ string_of_expr symbols e ^ ")\n"
^ string_of_stmt symbols s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr symbols e ^ ")\n" ^
      string_of_stmt symbols s1 ^ "else\n" ^ string_of_stmt symbols
s2
  | Ifelseif(e, s, eif_list, Block([])) ->
      let rec print_elseif eif =
        "else if(" ^ string_of_expr symbols eif.elseif_expr ^ ")\n" ^
        string_of_stmt symbols eif.elseif_stmt
      in
      "if (" ^ string_of_expr symbols e ^ ")\n" ^ string_of_stmt
symbols s ^
      String.concat "" (List.map print_elseif eif_list)
  | Ifelseif(e, s1, eif_list, s2) ->
      let rec print_elseif eif =
        "else if(" ^ string_of_expr symbols eif.elseif_expr ^ ")\n" ^
        string_of_stmt symbols eif.elseif_stmt
      in
      "if (" ^ string_of_expr symbols e ^ ")\n" ^ string_of_stmt
symbols s1 ^
      (String.concat "" (List.map print_elseif eif_list)) ^ "else\n"
^ string_of_stmt symbols s2
  | For(Assigna(v, e1, e2), s) ->
      "for (" ^ v ^ " = " ^ string_of_expr symbols e1 ^ " ; " ^ v ^ "
<= " ^ string_of_expr symbols e2 ^ " ; " ^
      v ^ "++)" ^ string_of_stmt symbols s
  | For(Assignb(v, e1, e2, e3), s) -> begin
      match e2 with
        Literal(l) ->
          if l = 0 then
            raise (Failure ("increment value cannot be zero"))
          else
            "for (" ^ v ^ " = " ^ string_of_expr symbols e1 ^ " ; " ^
v ^ " <= " ^ string_of_expr symbols e3 ^ " ; " ^
            v ^ " += " ^ string_of_expr symbols e2 ^ ")" ^
string_of_stmt symbols s
      | Floatlit(l) ->
          if l = 0.0 then
            raise (Failure ("increment value cannot be zero"))
          else
            "for (" ^ v ^ " = " ^ string_of_expr symbols e1 ^ " ; " ^
v ^ " <= " ^ string_of_expr symbols e3 ^ " ; " ^
            v ^ " += " ^ string_of_expr symbols e2 ^ ")" ^
string_of_stmt symbols s
      | Uminus(Literal(l)) ->
          if l = 0 then
            raise (Failure ("increment value cannot be zero"))
          else
            "for (" ^ v ^ " = " ^ string_of_expr symbols e1 ^ " ; " ^
v ^ " >= " ^ string_of_expr symbols e3 ^ " ; " ^
            v ^ " += " ^ string_of_expr symbols e2 ^ ")" ^
string_of_stmt symbols s
      | Uminus(Floatlit(l)) ->
          if l = 0.0 then
            raise (Failure ("increment value cannot be zero"))
          else
            "for (" ^ v ^ " = " ^ string_of_expr symbols e1 ^ " ; " ^
v ^ " >= " ^ string_of_expr symbols e3 ^ " ; " ^
            v ^ " += " ^ string_of_expr symbols e2 ^ ")" ^
string_of_stmt symbols s
      | _ -> raise (Failure ("increment value must be an integer or
float literal: " ^ string_of_expr symbols e2))
    end
  | While(e, s) -> "while (" ^ string_of_expr symbols e ^ ") " ^
string_of_stmt symbols s
  | KernelInit(k, l) ->
      let elist = List.fold_left expand_kernel_list [] l
      in
      let symbols_list = build_list (symbols, List.length elist, [])
      in
      let slist = List.map2 string_of_expr symbols_list elist
      in
      let x = string_of_int (List.hd (List.map (fun subl ->
List.length subl) l))
      in
      let y = string_of_int (List.length l)
      in
      k ^ " = new Kernel (" ^ x ^ ", "
                            ^ y ^ ", new float[]{" ^
                           (String.concat ", " slist) ^ "});\n"

let string_of_vdecl var = java_type_of var.vartype ^ " " ^
var.varname ^ " = " ^
  initial_value var.vartype ^ ";\n"

let string_of_global_vdecl var =
  "public static " ^ java_type_of var.vartype ^ " " ^ var.varname ^ "
= " ^
  initial_value var.vartype ^ ";\n"

(* declare return variable *)
let retvar_decl fdecl =
  if fdecl.rettype = "void" then ""
  else java_type_of fdecl.rettype ^ " " ^ fdecl.retname ^ " = " ^
  initial_value fdecl.rettype ^ ";\n"

(* return retname *)
let retvar_return fdecl =
  if fdecl.rettype = "void" then ""
  else "return " ^ fdecl.retname ^ ";\n"

let string_of_fdecl fdecl globals =
  (* let retname_list = build_list (fdecl.retname, List.length
fdecl.body, []) *)
  let symbols = globals @ fdecl.locals @ fdecl.formals @ [{ varname =
fdecl.retname; vartype = fdecl.rettype}]
  in
  let symbols_list = build_list (symbols, List.length fdecl.body, [])
  in
  fsignature fdecl ^ "\n{\n" ^
  retvar_decl fdecl ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  (* String.concat "" (List.map2 string_of_stmt retname_list
fdecl.body) ^ *)
  String.concat "" (List.map2 string_of_stmt symbols_list fdecl.body)
^
  retvar_return fdecl ^
  "}\n"

(*
let rec merge_vars_funcs vars funcs =
  if List.length funcs = 0 then vars
  else
    let func = List.hd funcs
    in
    { varname = func.fname; vartype = func.rettype } ::
merge_vars_funcs vars (List.tl funcs)
*)

let string_of_program (vars, funcs) =
  (* let globals = merge_vars_funcs vars funcs *)
  let global_funcs = List.map (fun a -> { varname = a.fname ^
"_global_func"; vartype = a.rettype }) funcs
  in
  let global_vars = List.map (fun a -> { varname = a.varname ^
"_global_var"; vartype = a.vartype }) vars
  in
  let globals = global_funcs @ global_vars (* concatenate global
variables and global functions to form symbols *)
  in
  let globals_list = build_list (globals, List.length funcs, [])
  in
  import_decl ^ class_decl ^
  String.concat "" (List.map string_of_global_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map2 string_of_fdecl funcs globals_list) ^
"\n}\n" ^
  "\n" ^ imgrender_decl
```

Matlip.ml

```
open Printf

let print = true

let file = "Matlip.java"

let _ =
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
(*  if print then *)
     Walker.string_of_program program;


    let oc = open_out file in
      let java = Javaprinter.string_of_program program in
        fprintf oc "%s" java;
        close_out oc

(*  else
    ignore (Interpret.run program)
*)
```

Matlap.java

```
package matlip;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.color.ColorSpace;
import java.awt.image.BufferedImage;
import java.awt.image.ColorConvertOp;
import java.awt.image.ColorModel;
import java.awt.image.ConvolveOp;
import java.awt.image.Kernel;
import java.awt.image.WritableRaster;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.RuntimeException;
import java.util.Hashtable;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 *
 * @author hendry
 */
public class Matlip {
   enum OPERATION { ADD,SUB,MUL,DIV }

     static BufferedImage imread(String path){
        try{
            BufferedImage  image;
            image = ImageIO.read(new File(path));
            return image;
        }
        catch (IOException e){
           System.out.println("Error:"+e.getMessage());
           return null;
        }
    }

    static void imshow(BufferedImage im){

        try {
            ImageRenderComponent irc = new ImageRenderComponent(im);
            irc.setOpaque(true);
            JFrame f = new JFrame("Display Image");
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setContentPane(irc);
            f.setSize(800,600);
            f.setVisible(true);
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }

    /*
     * the image formats supported by JRE are
     * BMP, bmp
       jpg, JPG
       wbm
       jpeg
       png, PNG
       JPEG
       WBMP
       GIF, gif
     */
    static void imsave(BufferedImage im,String path){
        try {
            File outputfile = new File(path);
            String format = path.substring(path.lastIndexOf(".")+1,
path.length());
            ImageIO.write(im, format, outputfile);
        }
        catch (IOException e) {
            System.out.println("Error:"+e.getMessage());
        }
    }

    static BufferedImage imnew(int width, int height, String type){
        BufferedImage im=null;

        if (type.contentEquals("RGB")){
            im = new BufferedImage(width, height,
BufferedImage.TYPE_3BYTE_BGR);
            for (int i=0;i<im.getHeight();i++){
                for (int j=0;j<im.getWidth();j++){
                    im.setRGB(j, i, 0xFF000000);
                }
            }

        }
        else if (type.contentEquals("GREY") ||
type.contentEquals("GRAY")){
            im = new BufferedImage(width, height,
BufferedImage.TYPE_BYTE_GRAY);
        }

        return im;
    }

    static Kernel kernelinit(){
        Kernel k = new Kernel(3,3,new float[] {
                    0.0f, 0.0f, 0.0f,
                    0.0f, 1.0f, 0.0f,
                    0.0f, 0.0f, 0.0f
                });
        return k;
    }

    static Kernel kernelnew(int width, int height){

        try{
            float[] data = new float[width*height];
            Kernel k = new Kernel(width, height, data);
            return k;
        }
        catch (Exception e){
            e.printStackTrace();
            return null;
        }
    }

    static void print(Object data){
        System.out.print(data);
    }


    static int getWidth(BufferedImage im){
        try {
            return im.getWidth();
        }
        catch (Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    static int getWidth(Kernel k){
        try {
            return k.getWidth();
        }
        catch (Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    static int getHeight(BufferedImage im){
        try {
            return im.getHeight();
        }
        catch (Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    static int getHeight(Kernel k){
        try {
            return k.getHeight();
        }
        catch (Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    static String getType(BufferedImage im){

        try{
            if (im.getType() == BufferedImage.TYPE_INT_ARGB ||
im.getType() == BufferedImage.TYPE_3BYTE_BGR || im.getType() ==
BufferedImage.TYPE_BYTE_INDEXED){
                return "RGB";
            }
            else if (im.getType() == BufferedImage.TYPE_BYTE_GRAY) {
                return "GREY";
            }
            else {
                return "unknown type";
            }
        }
        catch (Exception e){
            e.printStackTrace();
            return "ERROR: getType()";
        }
    }

    static int getLength(String s){
        try {
            return s.length();
        }
        catch (Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    static float getKernelData(Kernel k,int i, int j){
        float[] data = new float[k.getHeight()*k.getWidth()];
        k.getKernelData(data);
        return data[i+j*k.getWidth()];
    }

    static Kernel setKernelData(Kernel k, int i, int j, float value){
        float[] data = new float[k.getHeight()*k.getWidth()];
        k.getKernelData(data);
        data[i+j*k.getWidth()] = value;
        k = new Kernel(k.getWidth(),k.getHeight(),data);
        return k;
    }

   private static int objectToInt(Object o){
       if (o.getClass().getName().contentEquals("java.lang.Float")){
           return (int)Float.parseFloat(o.toString());
       }
       else if
(o.getClass().getName().contentEquals("java.lang.Double")){
           return (int)Double.parseDouble(o.toString());
       }
       else if
(o.getClass().getName().contentEquals("java.lang.Integer")){
           return Integer.parseInt(o.toString());
       }
       else if
(o.getClass().getName().contentEquals("java.lang.String")){
           return Integer.parseInt(o.toString());
       }
       else {
           throw new RuntimeException("type error!");
       }

   }

   private static float objectToFloat(Object o){
       if (o.getClass().getName().contentEquals("java.lang.Float")){
           return Float.parseFloat(o.toString());
       }
       else if
(o.getClass().getName().contentEquals("java.lang.Double")){
           return (float)Double.parseDouble(o.toString());
       }
       else if
(o.getClass().getName().contentEquals("java.lang.Integer")){
           return (float)Integer.parseInt(o.toString());
       }
       else if
(o.getClass().getName().contentEquals("java.lang.String")){
           return Float.parseFloat(o.toString());
       }
       else {
           throw new RuntimeException("type error!");
       }
   }
    static Object doArithmetic(Object op1, Object op2, OPERATION op){

            if (op1 instanceof BufferedImage){
```

```
        BufferedImage des;
        boolean color=false;
        try{
          int t =
((BufferedImage)op1).getSampleModel().getSample(0, 0, 1,
((BufferedImage)op1).getRaster().getDataBuffer());
          //des =
imnew(((BufferedImage)op1).getWidth(),((BufferedImage)op1).getHeight(
),"RGB");
          des = new
BufferedImage(((BufferedImage)op1).getWidth(),((BufferedImage)op1).ge
tHeight(),((BufferedImage)op1).getType());
          color=true;
        } catch (Exception e){
          des =
imnew(((BufferedImage)op1).getWidth(),((BufferedImage)op1).getHeight(
),"GREY");
        }

        if (op2 instanceof BufferedImage){
          if
(((BufferedImage)op1).getHeight()!=((BufferedImage)op2).getHeight()
|| ((BufferedImage)op1).getWidth()!=((BufferedImage)op2).getWidth())
            throw new RuntimeException("dimension
mismatch!");
        }



        int v1;
        int v2;
        for (int j=0;j<((BufferedImage)des).getHeight();j++){
          for (int
i=0;i<((BufferedImage)des).getWidth();i++){
            if (color){
              v1 = ((BufferedImage)op1).getRGB(i, j);
              if (op2 instanceof BufferedImage)
                v2 = ((BufferedImage)op2).getRGB(i,
j);
              else
                v2 = objectToInt(op2);


            }
            else{
              v1 =
((BufferedImage)op1).getSampleModel().getSample(i,j,0,((BufferedImage
)op1).getRaster().getDataBuffer());
              if (op2 instanceof BufferedImage)
                v2 =
((BufferedImage)op2).getSampleModel().getSample(i,j,0,((BufferedImage
)op2).getRaster().getDataBuffer());
              else
                v2 = objectToInt(op2);
            }

            switch (op){
              case ADD:
                if (color){
                  ((BufferedImage)des).setRGB(i, j,
(int)v1+v2);
```

```
                                  break;
                              }
                              else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1+v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                  break;
                              }
                         case SUB:
                             if (color){
                                  ((BufferedImage)des).setRGB(i, j,
(int)v1-v2);
                                  break;
                              }
                              else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1-v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                  break;
                              }
                         case MUL:
                              if (color){
                                  ((BufferedImage)des).setRGB(i, j,
(int)v1*v2);
                                  break;
                              }
                              else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1*v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                  break;
                              }
                         case DIV:
                             if (color){
                                  ((BufferedImage)des).setRGB(i, j,
(int)v1/v2);
                                  break;
                              }
                              else{

((BufferedImage)des).getSampleModel().setSample(i, j, 0,(int)v1/v2 ,
((BufferedImage)des).getRaster().getDataBuffer());
                                  break;
                              }
                    }
//                       if
(!((BufferedImage)des).isAlphaPremultiplied())
//
((BufferedImage)des).getSampleModel().setSample(i, j, 3,0xFF ,
((BufferedImage)des).getRaster().getDataBuffer());
                  }
              } // end of for
              return des;
         }

         else if (op1 instanceof Kernel){
              float[] des = new
float[((Kernel)op1).getWidth()*((Kernel)op1).getHeight()];
              float[] data1 = new
float[((Kernel)op1).getWidth()*((Kernel)op1).getHeight()];
              ((Kernel)op1).getKernelData(data1);
```

```
                float[] data2 = new
float[((Kernel)op1).getWidth()*((Kernel)op1).getHeight()];

                if (op2 instanceof Kernel){
                    if
(((Kernel)op1).getHeight()!=((Kernel)op2).getHeight() ||
((Kernel)op1).getWidth()!=((Kernel)op2).getWidth())
                        throw new RuntimeException("dimension
mismatch!");


                    ((Kernel)op2).getKernelData(data2);
                }


                float v1;
                float v2;

                for (int i=0;i<des.length;i++){
                    v1 = data1[i];
                    if (op2 instanceof Kernel)
                        v2 = data2[i];
                    else
                        v2 = objectToFloat(op2);

                      switch (op){
                            case ADD:
                               des[i] = v1+v2;
                               break;
                            case SUB:
                                des[i] = v1-v2;
                                break;
                            case MUL:
                                  des[i] = v1*v2;
                                  break;
                            case DIV:
                              des[i] = v1/v2;
                               break;
                        }
                }

                return new
Kernel(((Kernel)op1).getWidth(),((Kernel)op1).getHeight(),des);
            }
            else{
                throw new RuntimeException("op1 type error!");
            }
    }

    static Object clone(Object src){
        if (src instanceof BufferedImage){
             String[] pnames =
((BufferedImage)src).getPropertyNames();
             Hashtable<String, Object> cproperties = new
Hashtable<String,Object>();
             if(pnames != null) {
               for(int i = 0; i < pnames.length; i++) {
                   cproperties.put(pnames[i],
((BufferedImage)src).getProperty(pnames[i]));
               }
            }
            WritableRaster wr = ((BufferedImage)src).getRaster();
            WritableRaster cwr = wr.createCompatibleWritableRaster();
            cwr.setRect(wr);

            BufferedImage des = new BufferedImage(
               ((BufferedImage)src).getColorModel(),  // should be 
immutable
               cwr,
               ((BufferedImage)src).isAlphaPremultiplied(),
               cproperties);

            return des;


        }
        else if (src instanceof Kernel){
            return ((Kernel)src).clone();
        }
        else{
            throw new RuntimeException("the type of source is not 
cloneable!");
        }
    }

    static String read(){
        try{
            return (new BufferedReader(new 
InputStreamReader(System.in))).readLine();
        }
        catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }

    static void setImagePixel(BufferedImage im, int col, int row, 
String channel, int value){
        try{
            if (channel.toUpperCase().contentEquals("RGB")){
                    im.setRGB(col, row, value|0xFF000000);
            }
            else if (channel.toUpperCase().contentEquals("R")){
                    im.getSampleModel().setSample(col, row, 0, value, 
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals("G")){
                    im.getSampleModel().setSample(col, row, 1, value, 
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals("B")){
                    im.getSampleModel().setSample(col, row, 2, value, 
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals("GREY") || 
channel.toUpperCase().contentEquals("GRAY")){
                    im.getSampleModel().setSample(col, row, 0, value, 
im.getRaster().getDataBuffer());
            }
            else{
                throw new RuntimeException("invalid channnel!");
            }
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }

    static int getImagePixel(BufferedImage im, int col, int row,
String channel){
        try{
            if (channel.toUpperCase().contentEquals("RGB")){
                return im.getRGB(col, row)&0x00FFFFFF;
            }
            else if (channel.toUpperCase().contentEquals("R")){
                    return im.getSampleModel().getSample(col, row, 0,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals("G")){
                    return im.getSampleModel().getSample(col, row, 1,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals("B")){
                    return im.getSampleModel().getSample(col, row, 2,
im.getRaster().getDataBuffer());
            }
            else if (channel.toUpperCase().contentEquals("GREY") ||
channel.toUpperCase().contentEquals("GRAY")){
                    return im.getSampleModel().getSample(col, row, 0,
im.getRaster().getDataBuffer());
            }
            else{
                throw new RuntimeException("invalid channnel!");
            }
        }
        catch(Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    static BufferedImage convolve(BufferedImage im, Kernel k){
        BufferedImage imc = new BufferedImage(im.getWidth(),
im.getHeight(), im.getType());
        ConvolveOp convo = new ConvolveOp(k);
        convo.filter(im, imc);
        return imc;
    }
    public static void main(String[] args) {

           Kernel k = new Kernel(3,3,new float[] {
                            0.0f, -1.0f, 0.0f,
                            -1.0f, 5.0f, -1.0f,
                            0.0f,  -1.0f, 0.0f
                        });

        //TYPE_3BYTE_BGR   BMP JPG
        //TYPE_BYTE_INDEXED  GIF
        BufferedImage im =
imread("/home/hendry/Documents/PLT/project/matlip/rabbit.jpg");
       //
imsave(im,"/home/hendry/Documents/PLT/project/matlip/tmp.bmp");
```

```
       // im =
imread("/home/hendry/Documents/PLT/project/matlip/tmp.bmp");
       im = (BufferedImage)doArithmetic(im,1,OPERATION.ADD);

imsave(im,"/home/hendry/Documents/PLT/project/matlip/rabbit5.jpg");
        imshow(im);
        BufferedImage mulim =
(BufferedImage)doArithmetic(im,0,OPERATION.MUL);
        int a = getImagePixel(mulim,0,0,"RGB");
        System.out.println(a);
       // BufferedImage imm = convolve(im,k);
        //System.out.println(im.getType());
        //System.out.println(im.TYPE_3BYTE_BGR);
        //System.out.println(im.TYPE_BYTE_INDEXED);

        //imshow(imm);
        //System.out.println(getImagePixel(im,0,0,"G"));


       // imshow(mulim);

//        BufferedImage im_grey = (new
ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY),null)).filt
er(im, null);
//        imshow(im_grey);
//        BufferedImage ro =
imnew(im.getHeight(),im.getWidth(),"RGB");
//
//        for (int j=0;j<im.getHeight();j++){
//            for (int i=0;i<im.getWidth();i++){
//                ro.setRGB(j, i, im.getRGB(i, j));
//            }
//        }
      //  imshow(ro);



//        String s = "11";
//
//        int ii = objectToInt(s);
//        float ff = objectToFloat(ii);
//
//        System.out.println(ff);
//        System.out.println(ii);
//        float aa = 1.1f%2.2f;
//
//        String in = read();
//        System.out.println(in);
//        int a =10;
//        float b = 10f;
        //float aaaa = Math.sqrt(b);



//        BufferedImage imc = (BufferedImage)clone(im);
     //   imshow(im);
       // imshow(imc);
        //im = imnew(100,100,"RGB");
        //imshow(im);
        //imshow(imc);
        //BufferedImage sharpen = new
ConvolveOp(k,ConvolveOp.EDGE_NO_OP,null).filter(im, null);
        //imshow(sharpen);

//
//        Kernel k = new Kernel(3,3,new float[] {
//                    0.0f, -1.0f, 0.0f,
//                    -1.0f, 5.0f, -1.0f,
//                    0.0f, -1.0f, 0.0f
//                });
//         Kernel k1 = new Kernel(3,3,new float[] {
//                    0.0f, -1.0f, 0.0f,
//                    -1.0f, 5.0f, -1.0f,
//                    0.0f, -1.0f, 0.0f
//                });
        //Kernel kk = new Kernel(3,3,new float[] {});
        //Kernel k1 = k+3;

//        Kernel k2 = (Kernel)doArithmetic(k,k1,OPERATION.SUB);
//        float[] data = new float[k.getHeight()*k.getWidth()];
//        k2.getKernelData(data);
//
//        for (int i=0;i<data.length;i++)
//            System.out.print(data[i]+", ");



        //imshow(im);

        // kernel for sharpen
//        Kernel k = new Kernel(3,3,new float[] {
//                    0.0f, -1.0f, 0.0f,
//                    -1.0f, 5.0f, -1.0f,
//                    0.0f, -1.0f, 0.0f
//                });



       /**
        * syntax for declare kernel type
        * Kernel K = new Kernel(x,y,new float[]{v1,v2,....});
        */
//        Kernel k1 = new Kernel(3,3,new float[] {
//                    1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,
//                    1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,
//                    1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f
//                });
//

//        BufferedImage sharpen = new
ConvolveOp(k,ConvolveOp.EDGE_NO_OP,null).filter(im, null);
//        imshow(sharpen);
//
        /**
         * syntax for apply a kernel to an image
         * BufferedImage newIm = new
ConvolveOp(k1,ConvolveOp.EDGE_NO_OP,null).filter(im, null);
         * im: original image
         * k1: kernel
         * newIm: new image
         */
```

```
//        BufferedImage blur = new
ConvolveOp(k1,ConvolveOp.EDGE_NO_OP,null).filter(im, null);
//        imshow(blur);




    }

}


class ImageRenderComponent extends JPanel {
    BufferedImage image;
    Dimension size;

    public ImageRenderComponent(BufferedImage image) {
        this.image = image;
        size = new Dimension(image.getWidth(), image.getHeight());
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int x = (getWidth() - size.width)/2;
        int y = (getHeight() - size.height)/2;
        g.drawImage(image, x, y, this);
    }

    @Override
    public Dimension getPreferredSize() {
        return size;
    }
}
```

Myshell.sh

```
# bash
obj="./matlip"
log="maptip-test.log"
error=0

rm -f $log

Usage(){
      echo "Usage: myshell.sh [options] [.mp files]"
      echo "-h    Print this help"
      exit 1
}

Compare() {

    diff -b $1 $2  1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
      error=1
      echo "FAILED $1 differs from $2" 1>&2
    }
}

Run(){
      echo $* 1>&2
      eval $* || {
            echo "Test case failed on $*" 1>&2

            if [ $error -eq 0 ]
                  then error=1
            fi
            return 1
      }
}

Check(){
      casename=`echo $1 | sed 's/.*\///
                             s/.mp//'`
      reffile=`echo $1 | sed 's/.mp$/.out/'`

      echo 1>&2
      echo "========= Test Case
$casename===========================================" 1>&2


      outfile=$casename.out

      Run $obj "<" $1  &&
       javac Matlip.java 1>&2 &&
       java Matlip > $outfile
      Compare $outfile $reffile $casename.out.diff

      if [ $error -eq 0 ]
      then
            echo "PASSED!" 1>&2
      else
            echo "FAILED!" 1>&2

      fi
      rm -f $outfile $casename.out.diff
}

Fail(){
      casename=`echo $1 | sed 's/.*\///
                             s/.mp//'`
      reffile=`echo $1 | sed 's/.mp$/.out/'`

      echo 1>&2
      echo "========= Test Case
$casename===========================================" 1>&2


      #outfile=$casename.out

      Run $obj "<" $1 1>&2 &&
       javac Matlip.java  1>&2


      #Compare $casename.out $reffile $casename.out.diff
      #rm -f $outfile

}


while getopts h opt
```

```
do
      case $opt in
       h) # print usage
          Usage;;
      esac
done

# digest options
shift `expr $OPTIND - 1`


if [ $# -ge 1 ]
then  files=$@
else
      files="tests/*.mp"
fi

date>>$log
echo >>$log

for file in $files
do
     case $file in
      *test-*)
            Check $file 2>>$log;;
      *fail-*)
            Fail $file 2>>$log;;
      *)
            echo "$file is not a legal test file format";;
     esac
      error=0;
done

exit 0
```

Testfile

Test-for1-1.mp

```
function = main()
    int x;
    int i;
    x=3;
    for i=0:x:100
       x=x+1;
    end
end
```

fail-for1-2.mp

```
function = main()
    int x;
    int i;
    x=2;
    for i=0:(3+1):100
       x=x+1;
    end
end
```

fail-for1-3.mp

```
function int x= foo()
     x=3;
```

```
end
function = main()
    int x;
    int i;
    x=2;
    for i=0:foo():100
       x=x+1;
    end
end
```

fail-for1-4.mp

```
function = main()
    int i;
    for i=0:(int y):100
       x=x+1;
    end
    y=0;
end
```

fail-for2-1.mp

```
function int m = foo()
   m=3;
end


function = main()
    int x;
    int y;
    x=0;
    y=100;
    for x=x+foo():1:y=y+foo()
       x=x+1;
    end
end
```

fail-fun1-1.mp

```
#Error function is with three parameter but we call it with two
parameter
function int m = test (int x, int y, int z)

    m=x+y+z;

end

function = main ()

   int d;
   int x;
   int y;
   int z;
   x=1;
   y=2;
   z=3;
   d=test(x,y);

end
```

fial-fun1-2.mp

```
#Error function is with no parameter but we call it with one
parameter
function int m = test ()
```

```
    m=m+1;

end

function = main ()

   int d;
   int x;
   x=1;
   d=test(x);

end
```

fail-fun1-3.mp

```
#Error function is with two parameter but we call it with three
parameter

function int m = test (int x, int y)

    m=x+y;

end

function = main ()

   int d;
   int x;
   int y;
   int z;
   x=1;
   y=2;
   z=3;
   d=test(x,y,z);

end
```

fail-fun2-1.mp

```
#Error the parameter is of int type but we pass float into it

function int m = test (int x)

    m=x;

end

function = main ()

   int d;
   float x;
   x=1.0;
   d=test(x);

end
```

fail-fun3-1.mp

```
#Error Assign the function return int to a float variable
function int m = test (int x, int y, int z)

    m=x+y+z;

end

function = main ()
```

```
   float d;
   int x;
   int y;
   int z;
   x=1;
   y=2;
   z=3;
   d=test(x,y,z);

end
```

fail-fun4-1.mp

```
#ERROR call a function without return type but pass it to a variable
function = test (int x, int y, int z)
    int m;
    m=x+y+z;

end

function = main ()

   int d;
   int x;
   int y;
   int z;
   x=1;
   y=2;
   z=3;
   d=test(x,y,z);

end
```

fial-fun6-1.mp

```
#ERROR! call a function without declaration
function = main ()


   int x;
   int y;
   int z;
   x=1;
   y=2;
   z=3;
   test(x,y,z);

end
```

fial-if1-1.mp

```
function = main()
int x;
int y;
x=3;
y=0;

    if x==4
      y=4;
    else
      y=-1;
    end
    else if x==5
      y=5;
    else if x==6
```

```
      y=6;
    else if x==3;
       if y==0
          y=y+0;
       else
          y=y+3;
       end
    end

print(y);

end
```

fail-image1-1.mp

```
#  x = imnew(-1,-1,"RGB");  argument in the imnew cannot be negative
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(x, "./rabbit2.jpg");
  x = imnew(-1,-1,"RGB");
  imshow(x);
end
```

fail-image1-2.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(x, "./rabbit2.jpg");
  x = imnew(300.0,300.0,"RGB");
  imshow(x);
end
```

fail-image1-3.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(x, "./rabbit2.jpg");
  x = imnew(300,300,"");
  imshow(x);
end
```

fail-image1-4.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(x, "./rabbit2.jpg");
  x = imnew(300,300," ");
  imshow(x);
end
```

fail-image1-5.mp

```
function = main()
  int x;
  imshow(x);
end
```

fail-image1-6.mp

```
function = main()
  image x;
  imshow(x);
  x = imread();
  imshow(x);
end
```

fail-image1-7.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(x);
  x = imnew(300,300,"RGB");
  imshow(x);
end
```

fail-image1-8.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(,"./rabbit.jpg");
  x = imnew(300,300,"RGB");
  imshow(x);
end
```

fail-image1-9.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(3,"./rabbit.jpg");
  x = imnew(300,300,"RGB");
  imshow(x);
end
```

fail-image1-10.mp

```
# "put /home/sl2937/plt/matlip/rabbit.jpg"+x where x is of image type
in the imread
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg"+x);
  imshow(x);
  imsave(x,"./rabbit.jpg");
  x = imnew(300,300,"RGB");
  imshow(x);
end
```

fail-image1-11.mp

```
# imshow( imsave(x,"/home/sl2937/plt/matlip/rabbit.jpg") ); where
imsave(x,"/home/sl2937/plt/matlip/rabbit.jpg") is not an image and
cannot be shown

function = main()
  image x;
  imshow( imsave(x,"./rabbit.jpg") );
  imshow(x);
end
```

fail-image1-12.mp

```
# imshow(x+y+z)+imshow(x+y+z);  cannot add two imageshow

function image result = baboo(image i)
  result = i;
end

function = main()
  image x;
  image y;
  image z;
  image a;

  x = imread("./rabbit.jpg");
  y = imread("./rabbit.jpg");
  z = imread("./rabbit.jpg");


  z=imshow(x+y+z)+imshow(x+y+z);

end
```

fail-image1-14.mp

```
# x*0 is not a boolean type

function image result = baboo(image i)
  result = i;
end

function = main()
  image x;
  image y;
  image z;
  image a;

  x = imread("./rabbit.jpg");
  y = imread("./rabbit.jpg");
  z = imread("./rabbit.jpg");

  if x*0
     imshow(x+y+z);
  end
end
```

fail-image1-15.mp

```
function image result = baboo(image i)
  result = i;
end

function = main()
  image x;
  image y;
  image z;
  image a;
  string p;
  p="good";

  x = imread("./rabbit.jpg");
  y = imread("./rabbit.jpg");
  z = imread("./rabbit.jpg");
  print(p+x);
end
```

fail-imageassi1-1.mp

```
function = main()
image x;
int i;
x=imread("./rabbit.jpg");
i=0;
x=x/i;
x=x mod i;
x=x^i;
print(x[1,1,"RGB"]);
end
```

fail-kenel1-1.mp

```
#  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7]+[0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7] ;
function = main()
  image im;
  image im2;
  kernel k1;
  kernel k2;

  float f;
  int i;
  int j;

  im = imread("/home/sl2937/PLT/MATLIP/kitten.gif");
  imshow(im);

  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7]+[0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7] ;



end
```

fail-kenel1-2.mp

```
#  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0]
lack one element
function = main()
  image im;
  image im2;
  kernel k1;
  kernel k2;

  float f;
  int i;
  int j;

  im = imread("./kitten.gif");
  imshow(im);

  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0];
  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

end
```

fail-kenel1-3.mp

```
#im2 = k2 @ k1;
function = main()
  image im2;
  kernel k1;
  kernel k2;

  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0,
0.0,7.7];
  k2 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0,
0.0,7.7];


  k2 = k2 @ k1;

end
```

fail-kenel1-4.mp

```
function = main()
  image im;
  image im2;
  kernel k1;
  kernel k2;

  float f;
  int i;
  int j;

  im = imread("./kitten.gif");
  imshow(im);

  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0,
0.0,7.7];
  k2 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0,
0.0,7.7];


  im2 @ k2= im @ k1;
  imshow(im2);

end
```

fail-kenel1-5.mp

```
# k1=k3+k2; the dimension of k3+k2 is not the same with k1   cannot
add together and assign to k1
function = main()
  kernel k1;
  kernel k2;
  kernel k3;
  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k2 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k3 = [2.0; 3.0];
  if(k1!=k3)
    print("no");
  end

  k1=k3+k2;
  k1=k3*k2;
  k1=k3-k2;
  k1=k3/k2;
 end
```

fail-namespace1-1.mp

```
int x;
int x;
function = main()

end
```

fail-namespace1-2.mp

```
function =test()
end
function = main()

end
function = test()
end
```

fail-string1-1.mp

```
string x;
string y;
function string m = foo(string x,string y)
        m=x+y;
end

function = main()
    x="abc";
    y="cde";
    print(foo(x,y)-x);
end
```

fail-string1-2.mp

```
string x;
string y;
function string m = foo(string x,string y)
        m=x+y;
end

function = main()
    x="abc";
    y="cde";
    print(foo(x,y)*x);
end
```

fail-string1-3.mp

```
string x;
string y;
function string m = foo(string x,string y)
        m=x+y;
end

function = main()
    x="abc";
    y="cde";
    print(foo(x,y)*x);
end
```

test-for1-1.mp

```
function = main()
  int x;
  int i;
  i=0;
  for i=100:-1:0
      x=x+1;
  end
  print(x);
end
```

test-for1.mc

```
function = main()

  int i;
  for i = 0:4
    print(i);
  end
  print(42);
end
```

test-for2.mc

```
function = main()

  float i;
  float j;


  for i=1.0:3.0
    for j=1.0:3.0
      print(i);
      print(j);
    end
  end

  print(42);
end
```

test-for3.mc

```
function = main()

  float i;
  float j;


  for i=0.1:0.1:0.3
    for j=0.1:0.1:0.3
      print(i);
      print(j);
    end
  end

  print(42);
end
```

test-for4.mc

```
function = main()
  float i;
  float x;

  for i=5.0:-0.5:1.0
    print(i);
  end

  print(42);
```

```
end
```

test-fun5-1.mp

```
# OK! call a function with return type without assinging it to other variable
function int m = test (int x, int y, int z)

    m=x+y+z;

end

function = main ()


   int x;
   int y;
   int z;
   x=1;
   y=2;
   z=3;
   print(test(x,y,z));

end
```

test-fun7-1.mp

```
# OK -1 nested recursive
function int m = foo (int x)
    if(x > 0)
        m=foo(x-1);
    else
        m=-1;
    end
end

function int m = bar (int x)
    m=x+1;
end

function = main()
    print(foo(bar(5)));
end
```

test-fun7-2.mp

```
# OK -1 nested recursive
function int m = foo (int x)
    if(x > 0)
        m=foo(x-1);
    else
        m=-1;
    end
end

function int m = bar (int x)
    m=x+1;
end

function = main()
    print(foo(bar(foo(5))));
end
```

test-fun7-3.mp

```
# OK -1 mutully nested recursive
function int m = foo (int x)
```

```
    if x > 0
        m=bar(x-1);
    else
        m=-1;
    end
end

function int m = bar (int x)
    if x > 0
        m=foo(x-1);
    else
        m=-1;
    end
end

function = main()
     print(foo(bar(5)));
end
```

test-func1.mc

```
function float answer = add(float a, float b)
  answer = a + b;
end

function = main()
  float a;
  a = add(39.0, 3.0);
  print(a);
end
```

test-func2.mc

```
int global;

function int answer = foo()
   answer = 1;
   answer = 2;
   answer = 3;
end

function = change_global()
   global = 77;
end

function = main()
   int i;
   i = foo();
   print(i);

   global = 33;
   change_global();
   print(global);
end
```

test-gets1.mc

```
function = main()
  image im;
  kernel k;
  string s;

  im = imread("/home/ph2249/plt/matlip/kitten.gif");
  k = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7];
  s = "this is a test";
  print("image type = " + gettype(im));
  print("image width = " + getwidth(im));
  print("image height = " + getheight(im));

  print("kernel width = " + getwidth(k));
  print("kernel height = " + getheight(k));

  print("string length = " + getlength(s));

end
```

test-global1.mc

```
int a;
int b;

printa()
{
  print(a);
}

printb()
{
  print(b);
}

incab()
{
  a = a + 1;
  b = b + 1;
}

main()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
}
```

test-if1-1.mp

```
function = main()
int x;
int y;
x=3;
y=4;
    if x==3
       x=x+1;
    else
       x=x-1;
    end
    print(x);
end
```

test-if1-2.mp

```
function = main()
int x;
int y;
x=3;
y=4;
    if x==3
       if y==4
          x=x+1;
       else
          x=x-1;
       end
    else
       if x==5
          x=x-1;
       end
    end
    print(x);
end
```

test-if1-3.mp

```
function = main()
int x;
int y;
x=3;
y=4;
    if x==3

    else
       if y==4
          x=x+1;
       else
          if y==4
            x=x+1;
              if y==5
                x=x+1;
              end
          else
            x=x-1;
          end
       end
    end
    print(x);
end
```

test-if1-4.mp

```
function = main()
int x;
int y;
x=3;
y=0;

    if x==4
      y=4;
    elseif x==5
      y=5;
    elseif x==6
      y=6;
    elseif x==3
      y=3;
    end

print(y);

end
```

test-if1-5.mp

```
function = main()
int x;
int y;
x=3;
y=0;

    if x==4
      y=4;
    elseif x==5
      y=5;
    elseif x==6
      y=6;
    elseif x==3
       if y==0
          y=y+0;
       else
          y=y+3;
       end
    end

print(y);

end
```

test-if1-6.mp

```
function = main()
int x;
int y;
x=3;
y=0;

    if x==4
      y=4;
    elseif x==5
      y=5;
    elseif x==6
      y=6;
    elseif x==3
      y=3;
    else
        if x==4
           y=4;
        elseif x==5
           y=5;
        end
    end

print(y);

end
```

test-if1-5.mc

```
function = main()
  int i;
  int j;
  i = 3;
  j = 6;

  if i > 2
    print(i);
    print(j);
  else
    print(0);
    print(0);
  end
end
```

test-image1-1.mp

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  x=x-1;
  imshow(x);
  imsave(x, "./rabbit4.gif");
  x = imnew(300,300,"RGB");
  imshow(x);
end
```

test-image1-2.mp

```
function image m=test(image i)
  m=i;
end
function = main()
  image x;
  x = imread("./rabbit.jpg");

#  imshow(test(x));
#  imsave(x,"./rabbit.jpg");
#  x = imnew(300,300,"RGB");
#  imshow(x);
end
```

test-image1-3.mp

```
function = main()
  imshow(imread("./rabbit.jpg"));
end
```

test-image1-5.mp

```
function = main()
  imsave(imread("./rabbit.jpg"),"./rabbit.jpg");
end
```

test-image1-6.mp

```
function = main()
   image x;
   x=imnew(300,300,"RGB");
 #  imsave(x,"./rabbit3.jpg");
   imsave(imnew(300,300,"RGB"),"./rabbit3.jpg");
end
```

test-image1-7.mp

```
function image result = baboo(image i)
  result = i;
end

function = main()
  image x;
  image y;
  image z;
  image a;

  x = imread("./rabbit.jpg");
  y = imread("./rabbit.jpg");
  z = imread("./rabbit.jpg");
  imshow(x+y+z);


  imshow(baboo(imread("./kitten.jpg")));

end
```

test-image1-8.mp

```
function image result = baboo(image i)
  result = i;
end

function = main()
  image x;
  image y;
  image z;
  image a;

  int i;
  i=0;
  x = imread("./rabbit.jpg");
  y = imread("./rabbit.jpg");
  z = imread("./rabbit.jpg");


  for i=0:1:2
      x=baboo(x)+baboo(x);
      y=baboo(x)-baboo(y);
      z=baboo(x)*baboo(y);
      a=baboo(x)/baboo(x);
  end

   imshow(x);
   imshow(y);
   imshow(z);
    imshow(a);

end
```

test-image1-9.mp

```
function = main()
  image x;
  image y;
  int i;
  int j;

  x = imread("./rabbit.jpg");
  y = x; #clone an image

  for i=1:80
    for j=1:80
      x[i,j,"r"] = 0;
    end
  end

  for i=81:160
    for j=81:160
      x[i,j,"g"] = 0;
    end
  end

  for i=161:240
    for j=161:240
      x[i,j,"b"] = 0;
    end
  end

  for i=241:320
    for j=241:320
      x[i,j,"rgb"] = 0;
    end
  end

  imshow(x);
  imshow(y);

end
```

test-image1-10.mp

```
image x;
function int m = setimage (int i, int j, string k)

    x[i,j,k]=0;
    m=0;
end


function = main()

  image y;
  int i;
  int j;
  string k;
  k="r";
  y=y*2;
  x = imread("./rabbit.jpg");
  y = x; #clone an image

  for i=1:80
    for j=1:80
      x[i,j,k] = 0;
 #     setimage(i,j,k);
    end
  end

  for i=81:160
    for j=81:160
      x[i,j,"g"] = x[i,j,"g"]*0;
    end
  end

  for i=161:240
    for j=161:240
      x[i,j,"b"] = 0;
    end
  end

  for i=241:320
    for j=241:320
      x[i,j,"rgb"] = x[i,j,"rgb"]*0;
    end
  end

  imshow(x);
  imshow(y);

end
```

test-image1-1.mc

```
function = main()
  image x;
  imshow(x);
  x = imread("./rabbit.jpg");
  imshow(x);
  imsave(x, "./rabbit2.jpg");
  x = imnew(300,300,"RGB");
  imshow(x);
end
```

test-image1-2.mc

```
function = main()
  image x;
  image y;
  int i;
  int j;
  int k;
  int l;

  print("==rabit image below==");

  x = imread("./rabbit.jpg");
  i = x[20,20,"R"];
  j = x[20,20,"G"];
  k = x[20,20,"B"];
  l = x[20,20,"RGB"];
  print(i);
  print(j);
  print(k);
  print(l);

  print("==black image below==");

  y = imnew(100,100,"RGB");
  i = y[20,20,"R"];
  j = y[20,20,"G"];
  k = y[20,20,"B"];
  l = y[20,20,"RGB"];
  print(i);
  print(j);
  print(k);
  print(l);
end
```

test-image1-3.mc

```
function = main()
  image x;
  image y;
  int i;
  int j;

  x = imread("./rabbit.jpg");
  y = x; #clone an image
  for i=1:80
    for j=1:80
      x[i,j,"r"] = 0;
    end
  end

  for i=81:160
    for j=81:160
      x[i,j,"g"] = 0;
    end
  end

  for i=161:240
    for j=161:240
      x[i,j,"b"] = 0;
    end
  end

  for i=241:320
    for j=241:320
      x[i,j,"rgb"] = 0;
    end
  end

  imshow(x);
  imshow(y);

end
```

test-image1-4.mc

```
function = main()
  image x;
  image y;
  image z;
  image w;
  image a;
  int i;
  int j;

  x = imnew(100,100,"rgb");
  for i=0:99
    for j=0:99
      x[20,20,"r"] = 255;
    end
  end
  imshow(x); #red

  y = imnew(100,100,"rgb");
  for i=0:99
    for j=0:99
      y[i,j,"b"] = 255;
    end
  end
  imshow(y); #blue

  z = imnew(100,100,"rgb");
  for i=0:99
    for j=0:99
      z[i,j,"g"] = 255;
    end
  end
  imshow(z); #green

  w = imnew(100,100,"rgb");
  imshow(w); #black

  a = x + y + z + w;
  imshow(a); #should be white

  x = imread("./rabbit.jpg");

  imshow(x);
  imshow(x+x);
  imshow(x-x);
  imshow(x*x);
  imshow(x/x);

  #imshow(x+100);
  #imshow(x-100);
  #imshow(x*1.5);
  #imshow(x/1.5);

end
```

test-image1-5.mc

```
function image result = baboo(image i)
  result = i;
end

function = main()
  image x;
  image y;
  image z;
  image a;
  image b;
  kernel k;

  x = imread("./rabbit.jpg");
  y = imread("./rabbit.jpg");
  z = imread("./rabbit.jpg");

  a = x + y + z + 100;

  #b = 30 + x;
  k = k * 1.5;

  imshow(a);

  a = imread("./kitten.jpg");
  imshow(baboo(a)+50);

end
```

test-imageassi1-1.mp

```
function = main()
image x;
#image y;
int i;
#x=imnew(100,100,"RGB");
x=imread("./rabbit.jpg");
```

```
#y=imread("./rabbit.jpg");
i=0;
print(x[1,1,"RGB"]);
#y=x+y;
#imshow(y);
print(x[1,1,"RGB"]);
#y=x-y;
#imshow(y);
#print(x[1,1,"RGB"]);
x=x*i;
#imshow(y);
print(x[1,1,"RGB"]);
end
```

test-imageblur.mp

```
function = main()
   image x;
   image y;
   kernel k;
   k= [0.25,0.0,0.25;0.0,0.0,0.0;0.25,0.0,0.25];
   x=imread("./rabbit.jpg");
   #x=togray(x);
   imshow(x);
   y=x@k@k@k@k@k@k@k;
   imshow(y);
   imsave(y,"./r3.gif");
end
```

test-image-edge-detection.mp

```
function = main()
  image a;
  image b;
  kernel k;
  int i;
  int j;
  k = [-5.0, 0.0, 0.0;
        0.0, 0.0, 0.0;
        0.0, 0.0, 5.0];

  a = imread("./lena_color.jpg");
  b = togray(a);
  imshow(b);
  b = b@k;

  for i=0:getheight(b)
    for j=0:getwidth(b)
      if b[j,i,"grey"] < 0
        b[j,i,"grey"] = -b[i,j,"grey"];
      end
    end
  end
  imshow(b);
imsave(b,"./lena_edge.jpg");
end
```

test-flip-horizontal-detection.mp

```
function image ret = flip(image im)
      int height;
      int width;
      int i;
      int j;

      height = getheight(im);
      width = getwidth(im);

      ret=imnew(width,height,"RGB");

      for j=0:height-1
            for i=0:width-1
                  ret[width-i-1,j,"rgb"] = im[i,j,"rgb"];
            end
      end

end


function = main()
   image x;
   image y;


   x=imread("./rabbit.jpg");

   imshow(x);
   y=flip(x);
   imshow(y);
   imsave(y,"./r2.gif");
end
```

test-flip-vertical.mp

```
function image ret = flip(image im)
      int height;
      int width;
      int i;
      int j;

      height = getheight(im);
      width = getwidth(im);

      ret=imnew(width,height,"RGB");

      for j=0:height-1
            for i=0:width-1
                  ret[i,height-j-1,"rgb"] = im[i,j,"rgb"];
            end
      end

end


function = main()
   image x;
   image y;


   x=imread("./rabbit.jpg");

   imshow(x);
   y=flip(x);
   imshow(y);
   imsave(y,"./r.jpg");

end
```

test-flip-inverse.mp

```
function = main()
   image x;
   int i;
   int j;
   int width;
   int height;
     x=imread("./rabbit.jpg");
   width=getwidth(x);
   height=getheight(x);

   imshow(x);

   for j=0:height-1
        for i=0:width-1
             x[i,j,"R"] =  255-x[i,j,"R"];
             x[i,j,"G"] =  255-x[i,j,"G"];
             x[i,j,"B"] =  255-x[i,j,"B"];
        end
   end
   imshow(x);
   imsave(x,"./r5.jpg");

end
```

test-image-rotate.mp

```
function image ret = rotate90(image im)
      int height;
      int width;
      int i;
      int j;

      height = getheight(im);
      width = getwidth(im);

      ret=imnew(height,width,"RGB");

      for j=0:height-1
            for i=0:width-1
                  ret[height-j-1,i,"rgb"] = im[i,j,"rgb"];
            end
      end

end


function = main()
   image x;
   image y;
   image z;
   image w;
   image v;


   x=imread("./rabbit.jpg");

   imshow(x);
   y=rotate90(x);
   imshow(y);
   imsave(y,"./r7.jpg");
   z=rotate90(y);
   imshow(z);
   w=rotate90(z);
   imshow(w);
   v=rotate90(w);
   imshow(v);

end
```

test-image-sharpen.mp

```
function = main()
   image x;
   image y;
   kernel k;
   k= [0.0,-1.0,0.0;-1.0,5.0,-1.0;0.0,-1.0,0.0];

   x=imread("./rabbit.jpg");
   imshow(x);
   y=x@k@k;
   imshow(y);
   imsave(y,"./r4.gif");
end
```

test-kernel1-1.mp

```
function = main()
  image im;
  image im2;
  kernel k;

  float f;
  int i;
  int j;

  im = imread("/home/sl2937/PLT/MATLIP/kitten.gif");
  imshow(im);

  k = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7];

  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

  im2 = im @ k @ k;
  imshow(im2);

end
```

test-kernel1-2.mp

```
function = main()
  image im;
  image im2;
  kernel k;

  float f;
  int i;
  int j;

  im = imread("/home/sl2937/PLT/MATLIP/kitten.gif");
  imshow(im);
  k = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0, 
7.7];

  k[0,0]=0.0* k[0,0];
  k[0,1]=-1.0* k[0,1];

  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

end
```

test-kernel1-3.mp

```
function = main()
  int i;
  float j;
  kernel k;
  image l;
  k*k;
  2*3;
  i*i;
  j*j;
  l*l;

end
```

test-kernel1-4.mp

```
function = main()
  image im;
  image im2;
  kernel k;

  float f;
  int i;
  int j;

  im = imread("/home/sl2937/PLT/MATLIP/kitten.gif");
  imshow(im);
  f=0.0;
  k = [f, -1.0, f, 7.7; -1.0, 5.0, -1.0, 7.7; f, -1.0, f, 7.7];

  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

  im2 = im @ k;
  imshow(im2);

end
```

test-kernel1-5.mp

```
function = main()
  image im;
  image im2;
  kernel k;

  float f;
  int i;
  int j;

  im = imread("/home/sl2937/PLT/MATLIP/kitten.gif");
  imshow(im);
  f=0.0;
  k = [f+f, -1.0, f*f, 7.7; -1.0, 5.0, -1.0, 7.7; f-f, -1.0, f, 7.7];

  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

  im2 = im @ k;
  imshow(im2);

end
```

test-kernel1-6.mp

```
function = main()
  int width;
  int height;
  kernel k1;
  kernel k2;
  int i;
  int j;
  k2 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k1 = [k2[0,0], -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, k1[1,0]];
  k2=k1;
  width=getwidth(k2);
  height=getheight(k2);
  print(width + " " + height);

#  print(k2[0,0]);
#  print(k2[1,0]);
#  print(k2[2,0]);
#  print(k2[3,0]);

#  print(k2[0,1]);
#  print(k2[1,1]);
#  print(k2[2,1]);
#  print(k2[3,1]);

  for i=0:1:height-1
     for j=0:1:width-1
         print(k2[j,i]+ " ");
     end
  end
end
```

test-kernel1-7.mp

```
function = main()
  int width;
  int height;
  kernel k1;
  kernel k2;
  kernel k3;
  int i;
  int j;
  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k2 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k3 = [2.0; 3.0];
  k3=k3*k3;
  k2=k1*2.0+k2;
  width=getwidth(k2);
  height=getheight(k2);
  print(width + " " + height);
  for i=0:1:height-1
     for j=0:1:width-1
         print(k2[j,i]+ " ");
     end
  end
  print(k3[0,0]);
  print(k3[0,1]);
end
```

test-kernel1-8.mp

```
function kernel m= addkernel(kernel k1,kernel k2)
      m=k2+k1;
end
function kernel m= multkernel(kernel k)
      m=k*k;
end


function = main()
  int width;
  int height;
  kernel k1;
  kernel k2;
  kernel k3;
  int i;
  int j;
  k1 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k2 = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, 0.0, 0.0,7.7];
  k3 = [2.0; 3.0];

  k3=multkernel(k3);


  k2= addkernel(k1,k2)*k1 + addkernel(k1,k2);
  width=getwidth(k2);
  height=getheight(k2);
  print(width + " " + height);
  for i=0:1:height-1
     for j=0:1:width-1
         print(k2[j,i]+ " ");
     end
  end
  print(k3[0,0]);
  print(k3[0,1]);
end
```

test-kernel1.mc

```
function = main()
  image im;
  image im2;
  kernel k;
  kernel k2;

  float f;
  int i;
  int j;

  im = imread("/home/ph2249/plt/matlip/kitten.gif");
  imshow(im);

  k = [0.0, -1.0, 0.0, 7.7; -1.0, 5.0, -1.0, 7.7; 0.0, -1.0, 0.0,
7.7];

  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

  im2 = im @ k;
  imshow(im2);

  #clone kernel
  k2 = k;

  k[0,1] = 99.9;
  k[1,1] = 99.9;
  k[2,1] = 99.9;

  print("------");
  for j=0:2
    for i=0:3
      print(k[i,j]);
    end
  end

  #should contain unmodified kernel
  print("------");
  for j=0:2
    for i=0:3
      print(k2[i,j]);
    end
  end

end
```

test-kernelassi.mc

```
function = main()
kernel x;
float i;
i=2.0;
print(x[1,1]);
x=x+i;
print(x[1,1]);
x=x-i;
print(x[1,1]);
x=x*i;
print(x[1,1]);
end
```

test-kernels.mc

```
function = main()
  image a;
  image b;
  kernel k;
  int i;
  int j;
  k = [-5.0, 0.0, 0.0;
        0.0, 0.0, 0.0;
        0.0, 0.0, 5.0];

  a = imread("./soccer.jpg");
  b = togray(a);
  imshow(b);
  b = b@k;

  for i=0:getheight(b)-1
    for j=0:getwidth(b)-1
      if b[j,i,"grey"] < 0
        b[j,i,"grey"] = -b[i,j,"grey"];
      end
    end
  end
  imshow(b);
end
```

test-mods.mc

```
function = main()
  int i;
  int j;
  float a;
  float b;

  i = 14;
  j = 3;
  a = 2.2;
  b = 3.3;

  print(mod(i,j));
  print(mod(a,b));
end
```

test-namespace1-1.mp

```
#global variable name is the same with global function name
int x;
function int m=x()
   m=2;
end
function = main()
      print(x);
      print(x());
end
```

test-namespace1-2.mp

```
#local variable is the parameter of the global function
function int x=x()
   x=2;
end
function = main()
      print(x());
end
```

test-namespace1-3.mp

```
#local variable is inside the scope of the global function
function int m=x()
  int x;
   x=2;
   m=x;
end
function = main()
```

```
      print(x());
end
```

test-namespace1-4.mp

```
#local variable is inside the scope of the global function
function int m=test()
  int x;
   x=2;
   m=x;
end
function = main()
   int test;
   test=7;
   print(test());
   print(test);
end
```

test-namespace1-5.mp

```

int x;
function int x=test()
  x=4;
end
function = main()

   print(x);
   print(test());

end
```

test-namespace1-6.mp

```
int x;
function int m=test()
  int x;
  x=3;
  m=x;
end
function = main()

   print(x);
   print(test());

end
```

test-namespace.mc

```
image x;
int bar;

function image im = foo()
  im = imread("./rabbit.jpg");
  bar = 22;
  print(bar); #should print 22
end

function int answer = bar()
  answer = 3;
end

function = main()
  int bar;
  int i;
  float x;
  int foo;
  foo = 88;
  bar = 33;
  x = 7.0;

  #print(x+3.0);
  foo();
  i = bar + 4;
  print(i); #should print 37
  print(foo); #should print 88
end
```

test-ops.mc

```
main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  print(1 == 2);
  print(1 == 1);
  print(99);
  print(1 != 2);
  print(1 != 1);
  print(99);
  print(1 < 2);
  print(2 < 1);
  print(99);
  print(1 <= 2);
  print(1 <= 1);
  print(2 <= 1);
  print(99);
  print(1 > 2);
  print(2 > 1);
  print(99);
  print(1 >= 2);
  print(1 >= 1);
  print(2 >= 1);
}
```

test-power.mc

```
function = main()
  int i;
  int j;
  int k;
  float a;
  float b;

  i = 4;
  j = 3;
  k = 2;
  a = 2.2;
  b = 3.3;
  print(i^j^k);
  print(a^b);
end
```

test-read1-1.mp

```
function = main()
   image x;
   int height;
   int width;
   width=toint(read());
   height=toint(read());
   x=imnew(width,height,"RGB");
   imshow(x);
end
```

test-scope1-3.mp

```
int x;
function int m=test()
m=x;
end
function int m=test2()
int x;
x=1;
m=test();
end
function =main()
print(test2());
end
```

test-sqrt.mc

```
function = main()
  int i;
  float f;

  i = 36;
  f = 64.0;

  print(sqrt(i));
  print(sqrt(f));
end
```

test-string1-1.mp

```
function = main()
    string x;
    x="abc";
    print(x);
end
```

test-string1-2.mp

```
function = main()
    string x;
    string y;
    x="abc";
    y="def";
    x=x+y;
    print(x);
end
```

test-string1-3.mp

```
string x;
string y;
function string m = foo(string x,string y)
        m=x+y;
end

function = main()
    x="abc";
    y="cde";
    print(foo(x,y));
end
```

test-string1-4.mp

```
function = main()
    string x;
    int y;
    y=3;
    x="abc";
    print(x+y);
end
```

test-string1.mc

```
function float answer = calculate(float f)
  answer = 1.1;
end

function = main()
  string s1;
  string s2;
  string s3;
  float i;
  i = 7.7;

  s1 = "string1 ";
  s2 = "string2 " + "string3 ";
  s3 = s1 + s2 + i + " ";

  print(s3 + calculate(i));
end
```

test-string2.mc

```
string s1;
string s2;

function string answer = add_string(string a, string b)
  answer = a + b + "beverly!";
end

function = main()
  s1 = "hello, ";
  s2 = "patrick and ";
  print(add_string(s1, s2));
end
```

test-to1.mc

```
function = main()
  int i;
  int j;
  int k;
  float a;
  string s;
  image im;
  image im2;
  i = 77;
  j = 88;

  s = tostring(i+j);
  a = tofloat(i+j);
  k = toint(77.88);

  print(s);
  print(a);
  print(k);

  im = imread("/home/ph2249/plt/matlip/rabbit.jpg");
  imshow(im);
  im2 = togray(im);
  imshow(im2);
```

```
end
```