YAIL – Yet Another Image-processing Language

COMS W4115: Programming Languages and Translators, Fall 2010

Final Report

PROJECT TEAM

Pranay Prabhakar: ppp2113
Andrew Kisch: aik2113
Uday Chandrasen: uc2124
Aniket Phatak: avp2110

INSTRUCTOR

PROF. STEPHEN EDWARDS
Table of Contents

1. Introduction ........................................................................................................................................... 4
   1.1 Motivation ......................................................................................................................................... 4
   1.2 Goal ................................................................................................................................................ 4

2. Language Tutorial ................................................................................................................................... 5
   2.1 The “Main” method. ...................................................................................................................... 5
   2.2 Variables ......................................................................................................................................... 5
   2.3 Iteration and Conditional Control Structures .............................................................................. 6
      2.3.1 Iteration ..................................................................................................................................... 6
      2.3.2 Condition Control .................................................................................................................. 6
   2.4 Operators ......................................................................................................................................... 7
   2.5 Images .............................................................................................................................................. 8
      2.5.1 Image declaration and initialization ..................................................................................... 8
      2.5.3 Defining your own filters ....................................................................................................... 10

3. Language Manual ................................................................................................................................... 12
   3.1 Lexical Conventions ...................................................................................................................... 12
      3.1.1 Comments .............................................................................................................................. 12
   3.2 Data Types ...................................................................................................................................... 12
      3.2.1 int ............................................................................................................................................... 12
      3.2.2 float .......................................................................................................................................... 12
      3.2.3 string ....................................................................................................................................... 12
      3.2.4 image  This is a two dimensional buffer of values. The default size is 800X800 pixels. ....... 13
      3.2.5 filter ....................................................................................................................................... 13
   3.3 User Defined .................................................................................................................................... 13
      3.3.1 Functions ................................................................................................................................. 13
3.4 Type Conversions ................................................................................................................. 13
3.5 Expressions .......................................................................................................................... 13
  3.5.1 Primary Expressions ........................................................................................................ 13
  3.5.2 Unary Operators .............................................................................................................. 13
  3.5.3 Multiplicative Operators ............................................................................................... 13
  3.5.4 Additive Operators ....................................................................................................... 14
  3.5.5 Logical Operators ........................................................................................................... 14
  3.5.7 Image Operators ............................................................................................................ 14
3.6 Statements ........................................................................................................................... 15
  3.6.1 Conditional Statements ............................................................................................... 15
  3.6.2 Iteration .......................................................................................................................... 15
  3.6.3 Compound Statements ................................................................................................. 16
3.7 Scope ................................................................................................................................... 16
  3.7.2 Static Scoping .................................................................................................................. 16
  3.7.3 Global vs. Local .............................................................................................................. 16
  3.7.4 Forward Declarations .................................................................................................... 16
4. Project Plan ............................................................................................................................ 17
  4.1 Process ............................................................................................................................... 17
  4.2 Coding conventions .......................................................................................................... 17
  4.3 Project Timeline ............................................................................................................... 18
  4.4 Roles and Responsibilities of Team Members ................................................................. 19
  4.5 Development Environment ............................................................................................. 19
  4.6 Project Log ......................................................................................................................... 19
5. Architectural Design .............................................................................................................. 21
  5.1 Data flow ............................................................................................................................ 21
  5.2 Structure and Interface ...................................................................................................... 21
  5.3 Credits ................................................................................................................................ 22
6. Test Plan .................................................................................................................................. 23
7. Lessons Learned .................................................................................................................... 24
8. Appendix ............................................................................................................................... 26
1. Introduction

1.1 Motivation
Image processing has been an integral part of computer science ever since signal processing and digitizing analog data became possible. Digitized images lend themselves perfectly to classical applications of Mathematical Transformations like Fourier, Laplace et al. The applications of this fascinating field include Medical Imaging, Computer Vision, Biometrics, Computer Animation, Digital Cameras, Remote Sensing, Entertainment etc. Also, the fields of Artificial Intelligence like pattern matching etc. have found a new application with images.

Image Processing involves many stages, right from image acquisition, sampling, storing to processing and rendering. We are motivated to work on that part of Image Processing which involves the application of various transforms and computations in order to have the desired effect on a digital image.

1.2 Goal
We propose to build a language which will have syntax similar to that of C. In addition it will support a host of operations frequently needed in general image processing application development. We choose to call it YAIL – Yet Another Image processing Language for reasons obvious. Formally, the goals we aim to achieve are –

1. Flat learning curve for YAIL
2. First class support for images and filters
3. Easy implementation of frequent operations.
4. Rich built-in support for image operations
2. Language Tutorial
Since YAIL is C based, a prior knowledge of C is of immense help in learning YAIL. A simple algorithm to start off with YAIL:

1. Open your favorite editor.
2. Start off with the function main().
3. Write YAIL code within this function. You may also create your own functions.
4. Save the file with the .YAIL extension.
5. Call the YAIL compiler on the target file.

2.1 The “Main” method.
Every YAIL program begins execution at a method called main(). The syntax is:
return_type main() { //body}.

Example:

```yail
int main(){
    print("Hello World");
    return 0;
}
```

In the above program, the main function has return type int. The function body begins as “{“ and ends at “}”
. The “print” function is a built-in function provided by YAIL. It supports redirection of its arguments to the standard output stream. The “return” statement tells the compiler that the function should give back the value following the “return” statement to the calling process, in our case, the operating system.

2.2 Variables.
Variables, like in C are names for locations in memory. The data types for variables supported by YAIL are: int, float, image and filter. int and float have their usual meaning as in C. image refers to a type which can hold digital images. The filter type refers to a 2 dimensional matrix of floating point numbers. This is the usual Kernel as defined in the image processing literature.

All variables must be declared before being used in a YAIL program. Declaring means telling the YAIL compiler that you want to use a particular variable with a particular name and type.

Example:

```yail
int main(){
    int i;
    i = 42;
    print(i);
    return 0;
}
```
In the above example the first statement “int i” inside main() is a variable declaration. It tells the compiler that the type of the variable is int and the name is “i”. The second statement “i = 42” assigns the value 42 to the variable i.

The general syntax for declaring a variable is: `variable_type variable_name;

The syntax for assigning a value to a variable is: `variable_name_1 = value;

You may also assign one variable to another, provided both have been declared before.

2.3 Iteration and Conditional Control Structures

2.3.1 Iteration

For iteration, YAIL provides the FOR loop construct like in C. The syntax for iteration is:

```c
for(expr1;expr2;expr3){ //body }
```

Here, expr1 is the start condition. It is typically the assignment of a loop control variable to some legal value. Expr2 is the check condition. Every time before executing the body of the loop, YAIL will check whether expr2 evaluates to a Boolean TRUE value, if it does the loop body will be executed, otherwise the control will skip the loop. Expr 3 is the expression which changes the loop control variable value.

Example:

```c
int main(){
    int i;
    for(i=0; i< 5; i=i+1)
    {
        print("Hello World");
    }
    return 0;
}
```

This example will print the string “Hello World” five times on the console.

2.3.2 Condition Control

The `if..else` construct is supported by YAIL for conditional branching. The general syntax is:

```c
if(expr1){
    //body1
}
else{
    //body 2
}
```
Here, if expr1 evaluates to a boolean TRUE, then body1 is executed, else body2 is executed. If and else can be nested to an arbitrary depth. An else statement has to be associated to the lexically closest if statement just as in C.

Example:

```c
int main(){
    int i;
    for(i=0; i< 5; i=i+1){
        if ( i > 3){
            print("Hello World");
        }
        else{
            print("Hello");
        }
    }
    return 0;
}
```

In this example, at every iteration of the for loop, YAIL will check whether the value of i exceeds 3. If it does, it will print “Hello World”, otherwise, it will print only “Hello”. The else can also be safely omitted if not required.

### 2.4 Operators

The operators supported by YAIL are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>Binary</td>
<td>a+b</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Binary</td>
<td>a-b</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Binary</td>
<td>a*b</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Binary</td>
<td>a/b</td>
</tr>
<tr>
<td>-</td>
<td>Negation</td>
<td>Unary</td>
<td>-a</td>
</tr>
<tr>
<td>!</td>
<td>Logical inversion</td>
<td>Unary</td>
<td>!Y</td>
</tr>
</tbody>
</table>

Here, “a” and “b” can be variables of type int and float only. “a” and “b” can also be int literals or float literals. Y on the other hand, is an expression which has a Boolean value.
2.5 Images

2.5.1 Image declaration and initialization.

*image* is a type in YAIL. Like every other variable, one must first declare the image variable before using it.

```c
int main(){
    image img;
    image tempImg;
    img = newImage("/home/SomeImage.jpg");
    tempImg = img;
    printImage(img);
    printImage(tempImg);
    return 0;
}
```

The above program creates a variable called *img* of type *image*. This variable is then loaded with an image on disk at location /home/SomeImage.jpg. The function *newImage()* is a built-in function provided by YAIL to read images from disk. One may also assign one image variable to another. Here, the variable tempImg has a copy of the image held by variable img and hence the program will output two identical images.

2.5.2 YAIL support for some image operations

Following are the in-built functions supported by YAIL for image operations:

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>newImage()</td>
<td>var = newImage(&lt;Path&gt;)</td>
<td>Gets a variable to point to the image on disk</td>
<td>image img;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>img = newImage(&quot;a.jpg&quot;);</td>
</tr>
<tr>
<td>flipImageVertical()</td>
<td>var = flipImageVertical(img);</td>
<td>Takes a mirror image along the horizontal axis</td>
<td>image flipped;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flipped = flipImageVertical(img);</td>
</tr>
<tr>
<td>flipImageHorizontal()</td>
<td>var = flipImageHorizontal(img);</td>
<td>Takes a mirror image along the vertical axis</td>
<td>image flipped;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flipped = flipImageHorizontal(img);</td>
</tr>
<tr>
<td>meanFilter()</td>
<td>var = meanFilter(img);</td>
<td>Convolves the argument image with an averaging filter.</td>
<td>image meanImg;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>meanImg = meanFilter(img);</td>
</tr>
<tr>
<td>rotate()</td>
<td>var = rotate(theta, img);</td>
<td>Rotates the argument image</td>
<td>image rotIm;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>int theta;</td>
</tr>
<tr>
<td>Function</td>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>sharpen()</td>
<td>var = sharpen(img);</td>
<td>Sharpens the argument image by convolving with a sharpening filter.</td>
<td></td>
</tr>
<tr>
<td>edgeDetection()</td>
<td>var = edgeDetection(img);</td>
<td>Convolves the argument image with an edge-detection filter.</td>
<td></td>
</tr>
<tr>
<td>deleteImage()</td>
<td>deleteImage(str);</td>
<td>Deletes the image at the location str</td>
<td></td>
</tr>
<tr>
<td>saveImage()</td>
<td>saveImage(str, img);</td>
<td>Saved the argument image at location str</td>
<td></td>
</tr>
<tr>
<td>getColor()</td>
<td>var = getColor(img, x, y);</td>
<td>Gets color value of coordinates(x,y) in image img</td>
<td></td>
</tr>
<tr>
<td>getRed(), getGreen,</td>
<td>var = getRed(var)/getGreen(var)/getBlue(var)/getAlpha(var)</td>
<td>Gets you the red/green/blue/alpha value in the color represented by integer var</td>
<td></td>
</tr>
<tr>
<td>getBlue(), getAlpha()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare()</td>
<td>var = compare(img1, img2);</td>
<td>Returns True if img1 is same as img2, else returns False</td>
<td></td>
</tr>
</tbody>
</table>

```java
int theta = 45;
rotIm = rotate(theta, img);
image var;
var = sharpen(img);
image ed;
ed = edgeDetection(img);
deleteImage("a.jpg");
saveImage("foo.jpg", img);
int x;
x = getColor(img, 2, 2);
int z;
z = getRed(x);
image im1;
image im2;
boolean b;
b = compare(im1, im2);
```
2.5.3 Defining your own filters
A filter in YAIL is like another datatype and all the rules of datatype declaration apply to it. Following is an example code for using filters.

```c
int main()
{
    image im;
    filter sobelY;
    int i;
    i = 0;
    im = newImage("/home/ppp2113/yail/edgy.jpg");
    sobelY = { -1.0,-2.0,-1.0; 0.0, 0.0, 0.0; 1.0,2.0,1.0};
    printImage(im);
    printImage(im # sobelY );
    return i;
}
```

We have declared an image im above. Along with that we have declared a filter called SobelY. This is the Sobel operator which is used to find the gradient of the image intensities. What we have implemented is the Y gradient operator.

The syntax for filter definition is:

`filter_name = {column_1;column_2;column_3}`. Here each column specified should be a vector of the same length.

After specifying the filter, the way to apply it is use the # operator with the image object, i.e. `image_name # filter_name`. This operation does a 2-dimensional convolution of the image with the filter and produces the output image.
Here, we can verify that the filter works as expected, by noticing that the vertical edges are not as prominent as the more horizontal ones, due to the nature of the Sobel operator for the Y gradient.
3. Language Manual

3.1 Lexical Conventions
The various kinds of tokens in YAIL are as mentioned below. The tokens are separated by one or more of the following: comments, spaces, tabs or newlines. The separators are otherwise ignored. The tokens do not form unique prefixes. This means that a token is the longest substring of characters that could possibly be legal in the grammar defined.

3.1.1 Comments
Multi-line comments only. Comments start with /* and end with */. Everything else within the comments is ignored.

3.1.2 Identifiers
An identifier is a name given for a type or function. It can be a sequence of one or more letters and/or digits. Identifiers must start with a letter, not a digit. Underscores and special characters are not permitted. Identifiers are case sensitive.

3.1.3 Keywords
YAIL reserves the following keywords for itself. They may not be used as identifier in any YAIL program: int, float, if, else, for, return, image

3.1.4 Constants
- Integer
  Integer constants consist of decimal integers as a signed sequence of one or more digits.
- Float
  Float constants consist of signed decimal real numbers with one and only one decimal-point. The integral part is optional but the fractional part is not.
- String
  String constants will be any sequence of characters within double quotes. To have a double quote within string use the escape sequence ‘\’ followed by the double quote.

3.2 Data Types
Types define the “kind” of data that can be understood by YAIL. YAIL bases the interpretation of the identifiers based on their types. The following types are supported

3.2.1 int
32-bit signed integers are denoted by type int. Range of integer constants is -2147483648 to 2147483647, defaulting to 0.

3.2.2 float
64-bit signed decimal point numbers are denoted by type float. Default is 0.0.

3.2.3 string
Character strings are denoted by type string. Default is empty string “""."
3.2.4 image
This is a two dimensional buffer of values. The default size is 800X800 pixels.

3.2.5 filter
NxM matrix of floats, representing an image filter. The default value is
\{1.0,1.0,1.0;1.0,1.0,1.0;1.0,1.0,1.0\}; Here the syntax is \{row1;row2;row3...;rown\} where each row has the
same number of elements separated by commas.

3.3 User Defined
Besides the basic types the user can also define functions.

3.3.1 Functions
All functions in YAIL will have the following declarations pattern:

\[\text{returntype functionname (args-list) \{} \text{statements;} \}\]

Here, returntype should be a valid YAIL type, functionname is a valid identifier, and args
list should be a valid comma separated list of YAIL types. The args-list can be empty. The
function body should contain a valid YAIL statement or a group of statements. If the returntype is
not specified then the function will not return any value. Specifying a return statement in the body of
such a function will give an error. Otherwise, a return statement is necessary for a function specifying a
return type.

An exception to that is the main() method. This method cannot have any return type.

3.4 Type Conversions
YAIL is strictly typed and does not support any explicit type casting. Mixed mode arithmetic is not
supported by YAIL.

3.5 Expressions

3.5.1 Primary Expressions
A primary expression can be an identifier, any of the constants defined above, an expression contained
in parentheses.

3.5.2 Unary Operators
YAIL supports one unary operator for negation. It is denoted by a ‘-’ sign before an expression which
negates the expression. Only int and float values can be negated. Negation of numeric values means
multiplying the value by -1.

3.5.3 Multiplicative Operators
The multiplicative operators *, /, and % group left to right.

- Multiplication
  \[expression \ast expression\]
The binary * operator indicates multiplication. An int can be multiplied with an int or a float. A float can be multiplied with a float or an int. A int and float multiplication results in a float value. No other combinations are allowed.

- Division
  The binary / operator indicates division. The same type considerations as for multiplication apply.

3.5.4 Additive Operators
The additive operators + and - group left to right. An int can be added to an int or float. A float can be added to a float and int. The result of addition/subtraction involving float and int is a float.

3.5.5 Logical Operators
‘&&’ corresponds to logical AND. ‘||’ corresponds to logical OR. ‘!’ corresponds to logical NOT. ‘&&’ and ‘||’ are binary operators whereas ‘!’ is a unary operator.

3.5.6 Relational Operators
==, !=, <, <= , >, >= are all binary operators.

3.5.7 Image Operators
There are several image operators defined in YAIL for quick and easy modification of images. These operators are listed below-

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>newImage()</td>
<td>var = newImage(&lt;Path&gt;)</td>
<td>Gets a variable to point to the image on disk</td>
<td>image img; img = newImage(&quot;a.jpg&quot;);</td>
</tr>
<tr>
<td>flipImageVertical()</td>
<td>var = flipImageVertical(img);</td>
<td>Takes a mirror image along the horizontal axis</td>
<td>image flipped; flipped = flipImageVertical(img);</td>
</tr>
<tr>
<td>flipImageHorizontal()</td>
<td>var = flipImageHorizontal(img);</td>
<td>Takes a mirror image along the vertical axis</td>
<td>image flipped; flipped = flipImageHorizontal(img);</td>
</tr>
<tr>
<td>meanFilter()</td>
<td>var = meanFilter(img);</td>
<td>Convolves the argument image with an averaging filter.</td>
<td>image meanImg; meanImg = meanFilter(img);</td>
</tr>
<tr>
<td>rotate()</td>
<td>var = rotate(theta, img);</td>
<td>Rotates the argument image anti-clockwise by theta degrees</td>
<td>image rotIm; int theta; int theta = 45; rotIm = rotate(theta,img);</td>
</tr>
<tr>
<td>sharpen()</td>
<td>var = sharpen(img);</td>
<td>Sharpens the argument image by convolving with a</td>
<td>image var; var = sharpen(img);</td>
</tr>
</tbody>
</table>
sharpening filter.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edgeDetection()</code></td>
<td>Convolves the argument image with an edge-detection filter.</td>
<td><code>var = edgeDetection(img); image ed; ed = edgeDetection(img);</code></td>
</tr>
<tr>
<td><code>deleteImage()</code></td>
<td>Deletes the image at the location str</td>
<td><code>deleteImage(str); deleteImage(“a.jpg”);</code></td>
</tr>
<tr>
<td><code>saveImage()</code></td>
<td>Saved the argument image at location str</td>
<td><code>saveImage(str,img); saveImage(“foo.jpg”,img);</code></td>
</tr>
<tr>
<td><code>getColor()</code></td>
<td>Gets color value of co-ordinates(x,y) in image img</td>
<td><code>int x; x = getColor(img, 2, 2);</code></td>
</tr>
<tr>
<td><code>getRed(), getGreen,</code> <code>getBlue(), getAlpha()</code></td>
<td>Gets you the red/green/blue/alpha value in the color represented by integer var</td>
<td><code>int z; z = getRed(x);</code></td>
</tr>
<tr>
<td><code>compare()</code></td>
<td>Returns True if img1 is same as img2, else returns False</td>
<td><code>image im1; boolean b; b = compare(im1,im2);</code></td>
</tr>
</tbody>
</table>

### 3.6 Statements

#### 3.6.1 Conditional Statements
Conditional statements evaluate conditions and direct control flow appropriately.

```plaintext
if (expression) statement-block
if (expression) statement-block else statement-block
```

#### 3.6.2 Iteration
A valid for statement form is:

```plaintext
for(expression-statement; expression-statement; expression-statement) {
    statement-block
}
```

The first statement is evaluated before the loop begins, the second expression is evaluated at the beginning of each iteration and, if false, ends loop execution. The third statement is evaluated at the end of each iteration. Each expression can be multiple expressions separated by commas.
3.6.3 Compound Statements
Nested statements are permitted, such that selection and iteration statements can appear inside of a statement block. All statement blocks must begin with an open bracket and end with a close bracket.

3.7 Scope

3.7.2 Static Scoping
YAIL 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 YAIL, the scope of a variable is the most immediately enclosing block, excluding any enclosed blocks where the variable has been re-declared.

3.7.3 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 variables 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.

3.7.4 Forward Declarations
YAIL requires forward declarations for variables and functions. That is, a variable needs to be declared before it can be referenced, and any function needs to be defined before it can be invoked. For example, YAIL generally prohibits the following and will throw an error:

```plaintext
float a;
float b;
float mean;
mean = func(a, b);
```

... 

In this case, the function `func()` needs to be defined before it is called.
4. Project Plan

4.1 Process

1. **Initial Days:** During the early days of the course, the four of us would meet once a week for two hours to discuss and debate ideas for implementing a new language. This continued till the proposal was drafted.

2. **LRM phase:** This was the phase when the entire team was on a learning curve of OCaml. All of us had ambitious ideas for the language, but they had to be tempered down in keeping with our timeline. In this phase, the four of us met every week on Monday after the PLT class for 2-4 hours.

3. **Code Design and Implementation:** By the end of mid-term we had a fairly clear idea of how our language compiler was going to get built. We divided ourselves into two teams of two members each. One team took over the part of designing the grammar: AST, Scanner and Parser, whereas the other took over the part of designing the AST walk component and the Java converter thereof. As the functionality was getting added to the code base, each developer wrote small test cases for each of the functionality.

In this phase, each sub-team met at least once a week for 3-4 hours. Also both the sub-teams communicated regularly on email and also met once every 2 weeks to get everyone on the same page.

4.2 Coding conventions

The coding style was based on the Allman Style. Following were the conventions followed for the Java code:

1. Every function opening brace starts on a new line after the function name, under the same column as the first character of the function name.
2. Every function close brace starts on a new line after the last statement in the function body.
3. Every iterative structure and conditional branching structure has their opening and closing braces governed by rules similar to rule 1 and 2 above.
4. All member class members have to have their name starting in small case letters
5. Scoping is indicated by a single tab of 4 spaces.

Following are the coding conventions for OCaml code:

1. Descriptive comments occur immediately before the block of code they are related to.
2. Scoping is indicated by single tab of 4 spaces.
3. Rest of the coding convention is guided by the MicroC example presented by Prof Edwards.
4.3 Project Timeline

YAIL

First Team Meeting
Sept 17, 2010

Proposal
Sept 24, 2010

OCaml Preparation

LRM
Oct 22, 2010

Final Submission
Dec 22, 2010

- Studying Java Image Handling
- Tablegen (Symbol Table Gen)
- Javagen (Java Code Gen)
- Integration Testing

- Unit Testing

- Bug Fixing
4.4 Roles and Responsibilities of Team Members

<table>
<thead>
<tr>
<th>Person</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pranay Prabhakar</td>
<td>Front - end lead</td>
</tr>
<tr>
<td>Uday Chandrasen</td>
<td>Java lead</td>
</tr>
<tr>
<td>Aniket Phatak</td>
<td>Back-end lead</td>
</tr>
<tr>
<td>Andrew Kisch</td>
<td>Test lead</td>
</tr>
</tbody>
</table>

4.5 Development Environment

2. OCaml compiler: Version 3.11.2
3. Gedit for ocaml
4. Eclipse Helios, for Java backend
5. javac 1.6.0

4.6 Project Log

<table>
<thead>
<tr>
<th>Rev</th>
<th>Scores</th>
<th>Commit log message</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>264f3bfcc0</td>
<td></td>
<td>many changes from Uday and Aniket</td>
<td>Today (13 hours ago)</td>
<td><a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
</tr>
<tr>
<td>b58b9e0e3b</td>
<td></td>
<td>test suiting</td>
<td>Yesterday (16 hours ago)</td>
<td>AndrewKisch<a href="mailto:aik2113@gmail.com">aik2113@gmail.com</a></td>
</tr>
<tr>
<td>7e0c2cb9a7</td>
<td></td>
<td>Cannot return a literal done</td>
<td>Yesterday (19 hours ago)</td>
<td><a href="mailto:phatakaniket@gmail.com">phatakaniket@gmail.com</a></td>
</tr>
<tr>
<td>a0ae1f6857</td>
<td></td>
<td>error handling and line number in parser done</td>
<td>Yesterday (20 hours ago)</td>
<td><a href="mailto:phatakaniket@gmail.com">phatakaniket@gmail.com</a></td>
</tr>
<tr>
<td>77563c9a7b</td>
<td></td>
<td>convolve+return+global tested</td>
<td>Yesterday (20 hours ago)</td>
<td><a href="mailto:phatakaniket@gmail.com">phatakaniket@gmail.com</a></td>
</tr>
<tr>
<td>a9c1f5174f</td>
<td></td>
<td>tester file working</td>
<td>Yesterday (20 hours ago)</td>
<td>AndrewKisch<a href="mailto:aik2113@gmail.com">aik2113@gmail.com</a></td>
</tr>
<tr>
<td>01b21954f9</td>
<td></td>
<td>Adding first draft of report</td>
<td>Yesterday (17 hours ago)</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
</tr>
<tr>
<td>400c68e58a</td>
<td></td>
<td>Adding support for filters</td>
<td>Yesterday (21 hours ago)</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
</tr>
<tr>
<td>b3b2cdd130</td>
<td></td>
<td>Fixed bug in parenthesized expressions. Changes to all files except scanner.</td>
<td>Dec19, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
</tr>
<tr>
<td>0c899929128</td>
<td></td>
<td>Pranay: Fixed bug in parenthesized expressions</td>
<td>Dec 18, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
</tr>
<tr>
<td>b3c32d30f6</td>
<td></td>
<td>added support for width, height, error messages, put color</td>
<td>Dec15, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
</tr>
<tr>
<td>Commit</td>
<td>Description</td>
<td>Date</td>
<td>Author</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------</td>
<td>------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>ff3afc3bc5</td>
<td>testcases</td>
<td>Dec 12, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>3d50dd5ed4</td>
<td>merging changes made by uday/aniket Andrews/myself</td>
<td>Dec 11, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>aaa9047247</td>
<td>final version of ast, scanner, parser</td>
<td>Dec 10, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>7c256f2c17</td>
<td>uday</td>
<td>Nov 30, 2010</td>
<td><a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>70cbe00763</td>
<td>adding support for unary ops, for loop, boolean ops</td>
<td>Nov 30, 2010</td>
<td><a href="mailto:pranay.prabhakar@gmail.com">pranay.prabhakar@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>3a19f8a6bb</td>
<td>new Make</td>
<td>Nov 30, 2010</td>
<td><a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>7a19cc341d</td>
<td>Code error reverting back</td>
<td>Nov 30, 2010</td>
<td>udaychandrasen<a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>9832fa5cb4</td>
<td>same commit as before</td>
<td>Nov 30, 2010</td>
<td>AndrewKisch<a href="mailto:aik2113@gmail.com">aik2113@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>57dec3ef3b</td>
<td>updated part of the walker and added color and filter to the scanner</td>
<td>Nov 25, 2010</td>
<td>AndrewKisch<a href="mailto:aik2113@gmail.com">aik2113@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>6c9a31b530</td>
<td>all methods pushed</td>
<td>Nov 23, 2010</td>
<td>udaychandrasen<a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>37fb9c21f3</td>
<td>change to newImage method</td>
<td>Nov 23, 2010</td>
<td>udaychandrasen<a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>07568706ce</td>
<td>Updated by Uday &amp; Aniket</td>
<td>Nov 23, 2010</td>
<td>udaychandrasen<a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>d6d8acf78f</td>
<td>Updated by Uday &amp; Aniket</td>
<td>Nov 22, 2010</td>
<td>udaychandrasen<a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>c46711370a</td>
<td>Written by Uday &amp; Aniket</td>
<td>Nov 21, 2010</td>
<td>udaychandrasen<a href="mailto:udaychandrasen@gmail.com">udaychandrasen@gmail.com</a></td>
<td></td>
</tr>
</tbody>
</table>
5. Architectural Design

5.1 Data flow

YAIL leverages the functionality provided by Java libraries to attain its goals. Structurally, the YAIL compiler’s data flow looks like the following:

![Data Flow Diagram]

5.2 Structure and Interface

1. **Scanner**: This part of the compiler is the lexical analysis part. It is realized by the file `scanner.mll`. The scanner relies on the token definitions of the Parser, to read the input file and convert the valid character groupings into tokens.

2. **Parser**: The parser is realized by the file `parser.mly`. The role of the parser is to consume the tokens generated by the Scanner and try and fit them in the grammar defined in the parser. The parser also uses the interface defined by the `ast.ml` to generate the ast nodes. The parser builds the Abstract Syntax Tree of the input program.

3. **TableGen (Table Generator)**: This goes through the AST output and generates Symbol Table. This symbol table is a list containing global, local variables and functions. This is used further by JavaGen to prepare the java code. This module is realized by the file `tablegen.ml`

4. **JavaGen (Java Generator)**: This is used to produce the Java code equivalent to the code written in YAIL. JavaGen has a string of predefined methods and the skeleton of the java code to be
printed out. The JavaGen, uses the Symbol table generated by the TableGen and goes through the `ast` from the front end to comprehend the java code. This code is added to the already written methods and classes in file to output a complete Java Code to functionally accomplish the motive of the Yail user. This file is realized by the file `javagen.ml`

5. **Driver:** This is the component which takes user inputs and calls the compiler on the user input to produce the corresponding java file. This component is realized by the file `yail.ml`.

6. **Run:** Since `yail.ml` produces a java file, we need to call the java compiler to get the desired output. We have implemented a simple shell script which calls the `Makefile` for YAIL, takes the input source code file and calls the Driver program executable with this source file. Once this generates the java file, the script calls the java compiler on it to produce the desired output.

5.3 Credits

<table>
<thead>
<tr>
<th>Member</th>
<th>Worked On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew Kisch</td>
<td>test framework, ast.ml, scanner.ml, parser.ml</td>
</tr>
<tr>
<td>Aniket Phatak</td>
<td>tablegen.ml, javagen.ml, fixed bugs in other components</td>
</tr>
<tr>
<td>Pranay Prabhakar</td>
<td>ast.ml, parser.mly, scanner.mll, documentation, fixed bugs in other components</td>
</tr>
<tr>
<td>Uday Chandrasen</td>
<td>tablegen.ml, javagen.ml, source control setup, fixed bugs in other components</td>
</tr>
</tbody>
</table>
6. Test Plan

The testing of YAIL was implemented in two manners – one for testing the output of images, and one for testing everything else. Our discussion begins with the latter. In testing the non-image-specific functionality of the language, we adopted the testing practices implemented in MicroC. Namely, we created a file, named `testall.sh`, which would run a battery of tests, and save the output to temporary files. For each one of our tests, we also created files that contained the correct output. Our testall.sh would run each of the test-files and compare, via `diff`, the output files from the execution of our test files against the expected output in the files we had created. It reports failure if a difference exists, otherwise it returns success. For the image-based tester files, the output, instead of being printed, is saved to a file called `output.txt`. That file is then compared, once again, via `diff`, to the image file we expect to get.

Utilizing this test suite has been very helpful to us throughout the creation of YAIL. Since we are all new to OCaml specifically, and functional programming in general, we were consistently introducing bugs and this enabled us to see what was breaking – key information for repairing those bugs.

The following is a list of the functionality tested by each test file:

Non-images:

- `andOrNot` – Functionality of &&, || and !, also of booleans.
- `bops` – Functionality of binary operators
- `compare` – Functionality of the `compare` method for getting the equivalence of two images
- `deleteImage` – Functionality of deleting an image file on disk
- `for` – Functionality of for loops
- `getColors` – Functionality of accessing a specific pixel of an image, the getting our integer representation of its alphaRGB value and then separating out those four values into individual numbers
- `global1` – Ensures static scoping works properly
- `ifElseIf` – Functionality of condition statements
- `string1` – Checks that int, floats and strings work properly

Images:

- `embossConvolve` – Creates a filter and convolves
- `flipImageHorizontal` – Functionality of the `flipImageHorizontal` method
- `flipImageVertical` – Functionality of the `flipImageVertical` method
- `makeHalfBlack` – Functionality of `makeColor` to create our own color and `putColor` to manually manipulate the color of pixels in an image
- `meanFilter` – Functionality of the `meanFilter`, the smoothing filter. Extrapolate for all predefined filters


7. Lessons Learned

7.1 Aniket
I had spent almost 6 years programming in various different languages before taking this course. Initially, after taking this course, it was tough time learning Ocaml, a new functional language, for the first time in life. Gradually, I started acknowledging its recursive power for writing code for a compiler. Today, at the end of the semester, I am happy to understand the holistic view of how languages are written and not just how to write code in a language. Now, I feel confident to learn any new language given to me in my future life. Also, I can figure out which programming language will suit better for specific purposes. This will help me to choose different programming language for different purposes and not just follow the herd and code in Java but try out others which may be better suited for me then.

7.2 Pranay
1. It is important to plan in advance when the complexity of the project is high.
2. If there is much ambiguity at the start, having a tentative plan and working steadily towards understanding the problem at hand eventually makes things clear.
3. Timely communication within a team is very important for the team to succeed as a whole.
4. Functional Programming is a very useful tool for problems which involve patterns and recursion naturally.

7.3 Andrew
As my first semester-long, computer science group project, working on YAIL has taught me many lessons. The first of these lessons is one I feel I must constantly relearn – finish work sooner rather than later. Every person works at a different pace, and sometimes this can lead to inefficient uses of time due to bad planning. Fortunately, this was not often the case, however, it did occur a few times. The converse of that also holds true. Namely, that sometimes I was beyond inundated with other work and my group members who were more available at that time, and vice versa so overall we kept the project in motion. Working in pairs and then combining proved to be a great strategy for this project because the front-end and back-end made for a very good division of labor. On another level, as an undergraduate born and raised in New York City, working with three Indian graduate students has given me wonderful exposure to a culture with which I was not very familiar. I am very thankful I took this course and would gladly take another course like it. I feel it has made me a better computer scientist.

7.4 Uday
I am an ‘Electronics and Telecommunication’ undergrad and am pursuing Masters in Computer Science at Columbia. And hence, it was very important for me to take a course where I learn the ins and outs of programming languages. As Prof. Edwards rightly said, having studied the course we will now be able to have a better understanding of any programming language we encounter and will be able to throw very apt and precise questions about the language.
Apart from this, it was actually a very nice end to learning Ocaml. In the beginning, I really did not like it and found it to be very vague. But as I started to work on the project, I slowly understood that Ocaml is actually a very nice language to code compilers.

Apart from the knowledge I gained from the course and the project, I now have a star addition to my resume - my own image processing language “YAIL”, which is quite nice. All in all, I really enjoyed the course, the lectures and working on the project.
8. Appendix

8.1 AST.ml

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

type expr =
    IntLiteral of int (* literal of type integer *)
| FloatLiteral of float (*literal of type float*)
| StringLiteral of string (*literal of type string*)
| Datatype of string
| Id of string
| Binop of expr * op * expr
| Assign of string * expr
| Call of string * expr list
| Uminus of expr
| Not of expr
| Bool of string
| Noexpr
| Paren of expr
| FilterAccess of string * expr * expr
| NewFilter of expr * expr
| AssignFilter of string * expr * expr * expr

type stmt =
    Block of stmt list
| Expr of expr
| If of expr * stmt * stmt (* If expr evaluates to 1, first stmt is executed, else the second one is executed *)
| For of expr * expr * expr * stmt
| FilterInit of string * expr list list
type var_decl = {
    varname : string;
    vartype : string;
}

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

type program = var_decl list * func_decl list

8.2 Parser.ml

%{ open Ast
    open Lexing
    let parse_error message =
        let error_position = Parsing.rhs_start_pos 1 in
        let line = error_position.pos_lnum in
        print_endline (message ^ " in line " ^ string_of_int line)
    %}

%token SEMI COLON LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN AND OR NOT CONVOLVE
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE FOR NEWFILTER
program:
/* nothing */ { [] }
| program vdecl { ($2 :: fst $1), snd $1 }
| program fdecl { fst $1, ($2 :: snd $1) }

fdecl:
DATATYPE ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RETURN ID SEMI RBRACE
{
    fname = $2;
    rettype = $1;
    retname=$10;
    formals = $4;
    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
} }
8.3 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
    [" \	\r\n"] { token lexbuf } (* Whitespace *)
| "/*"   { comment lexbuf } (* Comments *)
| "*/"   { comment lexbuf } (* Comments *)
| """    { process_string "" lexbuf } (*start processing the string lit by passing an empty string *)
| '('   { LPAREN }
| ')'   { RPAREN }
| '{'   { LBRACE }
| '}'   { RBRACE }
| '['   { LBRACK }
| ']'   { RBRACK }
| "true" as lxm { BOOL(lxm) } |
| "false" as lxm { BOOL(lxm) } |

| \[0-9]+ as lxm { INTLIT(int_of_string lxm) } |
| \[0-9]+\.+([0-9]+\+\-.)? as lxm { FLOATLIT(float_of_string lxm) } |
| \[a-zA-Z\][a-zA-Z0-9] as lxm { ID(lxm) } |
| eof { EOF } |
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) } |

and comment = parse "#/" { token lexbuf } |
| _ { comment lexbuf } |

and process_string str = parse "\n" { if (String.length str) > 0 then |
| "\n" { let pos = lexbuf.Lexing.lex_curr_p in |
| raise (Failure("Cannot have a newline in a string. Error in line #" ^ (string_of_int pos.Lexing.pos_lnum))) |
| _ { process_string (str^(Lexing.lexeme lexbuf)) lexbuf } |

| "\n" { let pos = lexbuf.Lexing.lex_curr_p in |
| raise (Failure("Cannot have a newline in a string. Error in line #" ^ (string_of_int pos.Lexing.pos_lnum))) |
| _ { process_string (str^(Lexing.lexeme lexbuf)) lexbuf } |
8.4 Tablegen.ml

open Ast

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

let string_of_program (variables, methodName) =

let functionGen = List.fold_left
  (fun methodName temp1 ->
    if Table.mem temp1.fname methodName then
      raise (Failure("Error 13: SymbolGen method already defined"))
    else
      Table.add temp1.fname temp1 methodName) Table.empty methodName
in

let rec string_of_fdecl globals fdecl parameters =

let rec javaGen globals locals fname = function
  IntLiteral(x) -> { varname = string_of_int x;vartype = "int" }
  FloatLiteral(x) -> { varname = string_of_float x;vartype = "float" }
  StringLiteral(x) -> { varname = x;vartype = "string" }
  Bool(x) -> { varname = x;vartype = "boolean" }
  | Id(x) ->
    if Table.mem x locals then
      Table.find x locals
    else if Table.mem x globals then
      Table.find x globals


else raise (Failure ("Error 29: SymbolGen Identifier " ^ x ^ " could not be found"))

| Uminus(x) -> let temp = javaGen globals locals fname x in
  |
  | if temp.vartype = "int" || temp.vartype = "float" then
  |
  |   temp
  |
  | else
  |
  |   raise (Failure ("Error 39 : SymbolGen Unary Operation not used on float or int "))

| Datatype(x) ->
  |
  | if x = "parseerror" then
  |
  |   raise ((Failure ("Error 32: SymbolGen parsing Error")))
  |
  | else { varname = "null"; vartype = x }

| Paren(x) -> let temp = javaGen globals locals fname x in

| Binop(leftChild, o, rightChild) ->
  |
  | let temp1 = javaGen globals locals fname leftChild in
  |
  | (match o with
  |
  |     Add | Sub ->
  |
  |       let temp2 = javaGen globals locals fname rightChild in
  |
  |         if (temp1.vartype <> temp2.vartype) || temp1.vartype = "image" || temp1.vartype = "filter" || temp2.vartype = "image" || temp2.vartype = "filter") then
  |
  |           raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  |
  |       else temp1
  |
  |
  | Equal | Neq ->
  |
  |       let temp2 = javaGen globals locals fname rightChild in
  |
  |         if temp1.vartype <> temp2.vartype then
  |
  |           raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  |
  |         else { varname = "null"; vartype = "boolean" }

| Mult | Div ->
let temp2 = javaGen globals locals fname rightChild in

  if (temp1.vartype = "string" || temp2.vartype = "string" || temp1.vartype = "image" || temp2.vartype = "image" || temp1.vartype = "filter" || temp2.vartype = "filter") then
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  else if temp1.vartype = "boolean" || temp2.vartype = "boolean" then
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  else if temp1.vartype <> temp2.vartype then
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  else temp1

| Less | Leq | Greater | Geq -> |

let temp2 = javaGen globals locals fname rightChild in

  if (temp1.vartype = "string" || temp2.vartype = "string" || temp1.vartype = "boolean" || temp2.vartype = "boolean" || temp1.vartype <> temp2.vartype) then
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  else { varname = "null"; vartype = "boolean" }

| And | Or -> |

let temp2 = javaGen globals locals fname rightChild in

  if temp1.vartype = "boolean" && temp2.vartype = "boolean" then
    temp1
  else
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))

| Convolve -> |

let temp2 = javaGen globals locals fname rightChild in

  if temp1.vartype = "image" && temp2.vartype = "filter" then
    temp1
  else
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))

| Not(x) -> let temp = javaGen globals locals fname x in

  if temp.vartype = "boolean" then
temp
else
  raise (Failure ("Error 46: SymbolGen Not Operator incorrectly used "))

| FilterAccess(identifier, x_value, y_value) ->
  let temp1 = javaGen globals locals fname x_value in
  let temp2 = javaGen globals locals fname y_value in
  if Table.mem identifier locals then
    let var = Table.find identifier locals in
    if var.vartype <> "filter" then
      raise (Failure ("Error 130: tablegen.ml Filter type expected"))
    else if temp2.vartype <> "int" || temp1.vartype <> "int" then
      raise (Failure ("Error 132: tablegen.ml Invalid datatype for indexes"))
    else { varname = "null";vartype = "float" }
  else if Table.mem identifier globals then
    let var = Table.find identifier globals in
    if var.vartype <> "filter" then
      raise (Failure ("Error 137: tablegen.ml Filter type expected "))
    else if temp2.vartype <> "int" && temp1.vartype <> "int" then
      raise (Failure ("Error 139: tablegen.ml Invalid datatype for indexes"))
    else { varname = "null";vartype = "float" }
  else raise (Failure ("Error 141: tablegen.ml invalid path"))

| NewFilter(x_value, y_value) ->
  let temp1 = javaGen globals locals fname x_value in
  let temp2 = javaGen globals locals fname y_value in
  if temp2.vartype <> "int" || temp1.vartype <> "int" then
    raise (Failure ("Error 139: tablegen.ml Invalid datatype for indexes"))
else

{ varname = "filternew", vartype = "filter" }

| Assign(identifier, expression) -> |
if Table.mem identifier locals then
  let variable1 = Table.find identifier locals in
  let variable2 = javaGen globals locals fname expression in
  if variable1.vartype <> variable2.vartype then
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  else variable1
else if Table.mem identifier globals then
  let variable1 = Table.find identifier globals in
  let variable2 = javaGen globals locals fname expression in
  if variable1.vartype <> variable2.vartype then
    raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
  else variable1
else
  raise (Failure ("Error 67: SymbolGen incompatible Datatype"))

| AssignFilter(identifier, x_value, y_value, value) -> |
let varvalue = javaGen globals locals fname value in
let varRow = javaGen globals locals fname x_value in
let varCol = javaGen globals locals fname y_value in
if varvalue.vartype <> "float" then
  raise (Failure ("Error 67: SymbolGen incompatible Datatype"))
else
  if Table.mem identifier locals then
    let var = Table.find identifier locals in
    if var.vartype <> "filter" then

raise (Failure ("Error 112: tablegen.ml Filter type expected "))
else if varCol.vartype <> "int" && varRow.vartype <> "int" then
  raise (Failure ("Error 113: tablegen.ml Invalid datatype for indexes "))
else { varname = "null"; vartype = "float" }
else if Table.mem identifier globals then
  let var = Table.find identifier locals in
  if var.vartype <> "filter" then
    raise (Failure ("Error 112: tablegen.ml Filter type expected "))
  else if varCol.vartype <> "int" && varRow.vartype <> "int" then
    raise (Failure ("Error 113: tablegen.ml Invalid datatype for indexes "))
  else { varname = "null"; vartype = "float" }
else raise (Failure ("Invalid Path"))

| Call("print", parameters) ->

  let parameters = List.fold_left
    (fun parameters temp1 ->
      let variable = javaGen globals locals fdecl.fname temp1 in
      variable :: parameters)
    [] parameters
  in

  if (List.length parameters) <> 1 then
    raise (Failure ("Print excepts only 1 argument"))
  else
    let param = List.hd parameters in
    if param.vartype <> "image" && param.vartype <> "filter" then
      { varname = "print" ; vartype = "void" }
    else
      raise (Failure ("Error 163: Incorrect type given to string "))
Call("newImage", parameters) ->

let parameters = List.fold_left
(fun parameters temp1 ->
let variable = javaGen globals locals fdecl.fname temp1 in
variable :: parameters) [] parameters
in
if (List.length parameters) <> 1 then
raise (Failure ("newImage: Incorrect number of parameters") )
else
{ varname = "newImage" ; vartype = "image" }

Call("getColor", parameters) ->

let parameters = List.fold_left
(fun parameters temp1 ->
let variable = javaGen globals locals fdecl.fname temp1 in
variable :: parameters) [] parameters
in
if (List.length parameters) <> 3 then
raise (Failure ("getColor: Incorrect number of parameters"))
else
{ varname = "getColor" ; vartype = "int" }

Call("printImage", parameters) ->

let parameters = List.fold_left
(fun parameters temp1 ->
let variable = javaGen globals locals fdecl.fname temp1 in
variable :: parameters) [] parameters
in
variable :: parameters) □ parameters

in
if (List.length parameters) <> 1 then
  raise (Failure ("printImage: Incorrect number of parameters"))
else
  { varname = "printImage" ; vartype = "void" } | Call("getAlpha", parameters) ->

let parameters = List.fold_left
  (fun parameters temp1 ->
   let variable = javaGen globals locals fdecl.fname temp1 in
     variable :: parameters) □ parameters
in
if (List.length parameters) <> 1 then
  raise (Failure ("getAlpha: Incorrect number of parameters"))
else
  { varname = "getAlpha" ; vartype = "int" } | Call("getRed", parameters) ->

let parameters = List.fold_left
  (fun parameters temp1 ->
   let variable = javaGen globals locals fdecl.fname temp1 in
     variable :: parameters) □ parameters
in
if (List.length parameters) <> 1 then
  raise (Failure ("getRed: Incorrect number of parameters"))
else
let parameters = List.fold_left
     (fun parameters temp1 ->
      let variable = javaGen globals locals fdecl.fname temp1 in
      variable :: parameters) [] parameters
in
if (List.length parameters) <> 1 then
  raise (Failure ("getBlue: Incorrect number of parameters"))
else
  { varname = "getBlue" ; vartype = "int" }

let parameters = List.fold_left
     (fun parameters temp1 ->
      let variable = javaGen globals locals fdecl.fname temp1 in
      variable :: parameters) [] parameters
in
if (List.length parameters) <> 1 then
  raise (Failure ("getGreen: Incorrect number of parameters"))
else
  { varname = "getGreen" ; vartype = "int" }

let parameters = List.fold_left
     (fun parameters temp1 ->
      let variable = javaGen globals locals fdecl.fname temp1 in
      variable :: parameters) [] parameters

in
if (List.length parameters) <> 1 then
    raise (Failure ("flipImageVertical: Incorrect number of parameters"))
else
    { varname = "flipImageVertical" ; vartype = "image" }

| Call("flipImageHorizontal", parameters) ->
let parameters = List.fold_left
    (fun parameters temp1 ->
        let variable = javaGen globals locals fdecl.fname temp1 in
            variable :: parameters) [] parameters
in
if (List.length parameters) <> 1 then
    raise (Failure ("flipImageHorizontal: Incorrect number of parameters"))
else
    { varname = "flipImageHorizontal" ; vartype = "image" }

| Call("saveImage", parameters) ->
let parameters = List.fold_left
    (fun parameters temp1 ->
        let variable = javaGen globals locals fdecl.fname temp1 in
            variable :: parameters) [] parameters
in
if (List.length parameters) <> 2 then
    raise (Failure ("saveImage: Incorrect number of parameters"))
else
    { varname = "saveImage" ; vartype = "void" }
let parameters = List.fold_left
    (fun parameters temp1 ->
        let variable = javaGen globals locals fdecl.fname temp1 in
        variable :: parameters) [] parameters
    in
if (List.length parameters) <> 1 then
    raise (Failure "meanFilter: Incorrect number of parameters")
else
    { varname = "meanFilter" ; vartype = "image" }

let parameters = List.fold_left
    (fun parameters temp1 ->
        let variable = javaGen globals locals fdecl.fname temp1 in
        variable :: parameters) [] parameters
    in
if (List.length parameters) <> 2 then
    raise (Failure "rotate: Incorrect number of parameters")
else
    { varname = "rotate" ; vartype = "image" }

let parameters = List.fold_left
    (fun parameters temp1 ->
        let variable = javaGen globals locals fdecl.fname temp1 in
        variable :: parameters) [] parameters
    in
if (List.length parameters) <> 1 then
    raise (Failure ("sharpen: Incorrect number of parameters"))
else
    { varname = "sharpen" ; vartype = "image" }

| Call("edgeDetection", parameters) -> |
| Let parameters = List.fold_left |
| (fun parameters temp1 -> |
|     let variable = javaGen globals locals fdecl.fname temp1 in |
|         variable :: parameters) □ parameters |
| in |
| if (List.length parameters) <> 1 then |
|     raise (Failure ("edgeDetection: Incorrect number of parameters")) |
| else |
|     { varname = "edgeDetection" ; vartype = "image" } |

<p>| Call(&quot;deleteImage&quot;, parameters) -&gt; |
| Let parameters = List.fold_left |
| (fun parameters temp1 -&gt; |
|     let variable = javaGen globals locals fdecl.fname temp1 in |
|         variable :: parameters) □ parameters |
| in |
| if (List.length parameters) &lt;&gt; 1 then |
|     raise (Failure (&quot;deleteImage: Incorrect number of parameters&quot;)) |
| else |
|     { varname = &quot;deleteImage&quot; ; vartype = &quot;void&quot; } |</p>
<table>
<thead>
<tr>
<th>Call(&quot;putColor&quot;, parameters) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>let parameters = List.fold_left</td>
</tr>
<tr>
<td>(fun parameters temp1 -&gt;</td>
</tr>
<tr>
<td>let variable = javaGen globals locals fdecl.fname temp1 in</td>
</tr>
<tr>
<td>variable :: parameters) □ parameters</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>if (List.length parameters) &lt;&gt; 4 then</td>
</tr>
<tr>
<td>raise (Failure (&quot;putColor: Incorrect number of parameters&quot;))</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>{ varname = &quot;putColor&quot; ; vartype = &quot;image&quot; }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call(&quot;compare&quot;, parameters) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>let parameters = List.fold_left</td>
</tr>
<tr>
<td>(fun parameters temp1 -&gt;</td>
</tr>
<tr>
<td>let variable = javaGen globals locals fdecl.fname temp1 in</td>
</tr>
<tr>
<td>variable :: parameters) □ parameters</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>if (List.length parameters) &lt;&gt; 2 then</td>
</tr>
<tr>
<td>raise (Failure (&quot;compare: Incorrect number of parameters&quot;))</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>{ varname = &quot;compare&quot; ; vartype = &quot;boolean&quot; }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call(&quot;makeColor&quot;, parameters) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>let parameters = List.fold_left</td>
</tr>
<tr>
<td>(fun parameters temp1 -&gt;</td>
</tr>
<tr>
<td>let variable = javaGen globals locals fdecl.fname temp1 in</td>
</tr>
<tr>
<td>variable :: parameters) □ parameters</td>
</tr>
<tr>
<td>in</td>
</tr>
</tbody>
</table>
if (List.length parameters) <> 4 then
  raise (Failure("makeColor: Incorrect number of parameters"))
else
  { varname = "makeColor" ; vartype = "int" }

| Call("getHeight", parameters) ->
  let parameters = List.fold_left
  (fun parameters temp1 ->
    let variable = javaGen globals locals fdecl.fname temp1 in
    variable :: parameters)
  [] parameters
  in
  if (List.length parameters) <> 1 then
    raise (Failure("getHeight: Incorrect number of parameters"))
  else
    { varname = "Height" ; vartype = "int" }

| Call("getWidth", parameters) ->
  let parameters = List.fold_left
  (fun parameters temp1 ->
    let variable = javaGen globals locals fdecl.fname temp1 in
    variable :: parameters)
  [] parameters
  in
  if (List.length parameters) <> 1 then
    raise (Failure("getHeight: Incorrect number of parameters"))
  else
    { varname = "getHeight" ; vartype = "int" }

| Call(f, parameters) ->
let fdecl =

  if f = "main" then
      raise (Failure ("Incorrect use of main call"))
  else
    try Table.find f functionGen
    with Not_found -> raise (Failure ("Method " ^ f ^ " not found"))
  in
  if f <> fname then
      let parameters = List.fold_left
        (fun parameters temp1 ->
         let variable = javaGen globals locals fdecl.fname temp1 in
          variable :: parameters) [] parameters
      in
      string_of_fdecl globals fdecl (List.rev parameters);
      { varname = "" ; vartype = fdecl.rettype }
  else
    let parameters = List.fold_left
      (fun parameters temp1 ->
         let variable = javaGen globals locals fdecl.fname temp1 in
          variable :: parameters) [] parameters
    in
    try List.iter2 (fun formal temp1 ->
        if formal.vartype <> temp1.vartype then
          raise (Failure ("incompatible data types ")))
        fdecl.formals (List.rev parameters)
    with Invalid_argument(_ _) ->
        raise (Failure ("incorrect no. of arguments passed to method "));
let checkDimension filter 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 size of filter"))
      else c1
    in loop l
  in

let checkFilterValues globals locals filter fname l =
  let f = List.concat l in
  List.iter
    (fun exp ->
      let variable = javaGen globals locals fname exp in
      if variable.vartype <> "float" then
        raise (Failure ("A filter accepts only float values ")) f
    ) f
  in

let consume e = ()

let rec codeGen globals locals fname = function
  Block(stmts) ->
    List.iter (codeGen globals locals fname) stmts
Expr(expr) -> let e = javaGen globals locals fname expr in consume e

For(expression1,expression2,expression3,statement) ->
    let variable1 = javaGen globals locals fname expression2 in
    if variable1.vartype <> "boolean" then
        raise (Failure("Bad use of for "))
    else
        codeGen globals locals fname statement;

If(expression, statement, Block([])) -> let variable = javaGen globals locals fname expression in
    if variable.vartype <> "boolean" then
        raise (Failure("Invalid type given to if"))
    else
        codeGen globals locals fname statement;

If(expression, statement1, statement2) ->
    let variable = javaGen globals locals fname expression in
    if variable.vartype <> "boolean" then
        raise (Failure("Invalid Type given to if"))
    else
        codeGen globals locals fname statement1;
        codeGen globals locals fname statement2;

FilterInit(identifier, valuesList) ->
    if Table.mem identifier locals then
        let var = Table.find identifier locals in
        if var.vartype <> "filter" then
            raise (Failure("Error 469: tablegen.ml invalid variable type"))
        else
            let col = checkDimension identifier fname valuesList in
            consume col;
            checkFilterValues globals locals identifier fname valuesList
else if Table.mem identifier globals then
  let var = Table.find identifier globals in
  if var.vartype <> "filter" then
    raise (Failure ("Error 477: tablegen.ml invalid variable type"))
  else
    let col = checkDimension identifier fname valuesList in
    consume col;
    checkFilterValues globals locals identifier fname valuesList
    else raise (Failure ("Error 482: tablegen.ml invalid use of code found"))

in
let locals =
  try List.fold_left2
  (fun locals formal temp1 ->
    if formal.vartype = temp1.vartype then
      if Table.mem formal.varname locals then
        raise (Failure ("Multiple definition of the same variable"))
      else Table.add formal.varname formal locals
      else raise (Failure ("Error 492: tablegen.ml Incapitibility found")))
  Table.empty fdecl.formals parameters
  with Invalid_argument(_) ->
    raise (Failure ("Error 495: Arguments passed is invalid "))
  in
let locals = List.fold_left
  (fun locals local ->
    if Table.mem local.varname locals then
      raise (Failure ("Multiple definition of the same variable "))
    else Table.add local.varname local locals)
  locals fdecl.locals
in
if false then
  if Table.mem fdecl.rettype locals then
    raise (Failure ("Multiple definition of the same variable "))
  else
    let ret = { varname = "" ; vartype = fdecl.rettype } in
    let locals = Table.add ret.varname ret locals in
    List.iter (codeGen globals locals fdecl.fname) fdecl.body
else
  List.iter (codeGen globals locals fdecl.fname) fdecl.body
in
let globals = List.fold_left
  (fun globals vdecl ->
    if Table.mem vdecl.varname globals then
      raise (Failure ("Multiple definition of the same variable "))
    else
      Table.add vdecl.varname vdecl globals)
Table.empty variables
in
try
  string_of_fdecl globals (Table.find "main" functionGen) []
with Not_found ->
  raise (Failure ("Main function is absent !"))

8.5 Javagen.ml

open Ast

let javaParameterIntailize yailVariable =
  if yailVariable = "int" then "0"
else if yailVariable = "float" then "(float)0.0"
else if yailVariable = "string" then "\\\n"
else if yailVariable = "filter" then "filterinit()"
else if yailVariable = "color" then "-12000"
else if yailVariable = "boolean" then "false"
else if yailVariable = "image" then "new BufferedImage(800, 800, BufferedImage.TYPE_3BYTE_BGR)"
else
    raise (Failure ("Error 278 : JavaGen " ^ yailVariable ^ " unrecognized"))

let javaParameterType yailVariable =
    if yailVariable = "int" then "int"
    else if yailVariable = "float" then "float"
    else if yailVariable = "string" then "String"
    else if yailVariable = "filter" then "Kernel"
    else if yailVariable = "boolean" then "boolean"
    else if yailVariable = "color" then "int" (* inform to other team !!!! *)
    else if yailVariable = "image" then "BufferedImage"
    else if yailVariable = "" then "void"
    else
        raise (Failure ("Error 292 : JavaGen " ^ yailVariable ^ " unrecognized"))

let yailMethodConversion methodName =
    if methodName = "print" then "System.out.println"
    else if methodName = "saveImage" then "saveImage"
    else if methodName = "getAlpha" then "getAlpha"
    else if methodName = "getRed" then "getRed"
    else if methodName = "getBlue" then "getBlue"
    else if methodName = "getGreen" then "getGreen"
else if methodName = "getColor" then "getColor"
else if methodName = "flipImageHorizontal" then "flipImageHorizontal"
else if methodName = "flipImageVertical" then "flipImageVertical"
else if methodName = "meanFilter" then "meanFilter"
else if methodName = "rotate" then "rotate"
else if methodName = "compare" then "compare"
else if methodName = "sharpen" then "sharpen"
else if methodName = "edgeDetection" then "edgeDetection"
else if methodName = "deleteImage" then "deleteImage"
else if methodName = "putColor" then "putColor"
else if methodName = "getBuff" then "getBuff"
else if methodName = "newImage" then "newImage"
else if methodName = "makeColor" then "makeColor"
else if methodName = "getWidth" then "getWidth"
else if methodName = "getHeight" then "getHeight"
else methodName

let rec expressionClassification symbols = function
    | IntLiteral(x) -> { varname = string_of_int x; vartype = "int" } |
    | FloatLiteral(x) -> { varname = string_of_float x; vartype = "float" } |
    | StringLiteral(x) -> { varname = x; vartype = "string" } |
    | Bool(x) -> { varname = x; vartype = "boolean" } |
    | Id(x) ->
        let checkLocalSymbol = List.exists (fun temp -> if temp.varname = x then true else false) symbols
        in
        if checkLocalSymbol then
            let local = List.find (fun temp -> if temp.varname = x then true else false) symbols
            in
            ...
local

else

let checkGlobalSymbol = List.exists (fun temp -> if temp.varname = x ^ "_global_var" then true else false) symbols

in

if checkGlobalSymbol then

let global = List.find (fun temp -> if temp.varname = x ^ "_global_var" then true else false) symbols

in

{ varname = x; vartype = global.vartype }

else

raise (Failure ("Error 375 JavaGen : Symbol not found"))

| Uminus(x) ->

    let temp = expressionClassification symbols x

    in temp

| Not(x) ->

    let temp = expressionClassification symbols x

    in temp

| Binop(leftChild, o, rightChild) -> begin

match o with

    Equal

        | Neq

        | Less

        | Leq

        | Greater

        | Geq ->

            { varname = "null"; vartype = "boolean" }

    | _ ->

        let temp = expressionClassification symbols leftChild
in temp
end

| NewFilter(x_value, y_value) -> { varname = "filternew"; vartype = "Kernel" } |
| Assign(identifier, expression) -> |

let checkLocalSymbol = List.exists (fun temp -> if temp.varname = identifier then true else false) symbols in
if checkLocalSymbol then
let variable = List.find (fun temp -> if temp.varname = identifier then true else false) symbols in
variable else
let checkGlobalSymbol = List.exists (fun temp -> if temp.varname = identifier ^ "_global_var" then true else false) symbols in
if checkGlobalSymbol then
let variable = List.find (fun temp -> if temp.varname = identifier ^ "_global_var" then true else false) symbols in
{ varname = identifier; vartype = variable.vartype } else
raise (Failure ("Error 409 : JavaGen " ^ identifier ^ " unrecognized")) |
| AssignFilter(identifier, x_value, y_value, value) -> { varname = "null"; vartype = "float" } |
| FilterAccess(identifier, x_value, y_value) -> { varname = "null"; vartype = "float" } |
| Call(methodName, expression) -> |

let checkGlobalSymbol = List.exists (fun temp -> if temp.varname = methodName ^ "_global_func" then true else false) symbols in
if checkGlobalSymbol then
let temp1 = List.find (fun temp -> if temp.varname = methodName ^ "_global_func" then true else false) symbols
in
  { varname = methodName; vartype = temp1.vartype }
else if methodName = "getAlpha" then
  { varname = "getAlpha"; vartype = "int" }
else if methodName = "getRed" then
  { varname = "getRed"; vartype = "int" }
else if methodName = "getGreen" then
  { varname = "getGreen"; vartype = "int" }
else if methodName = "getBlue" then
  { varname = "getBlue"; vartype = "int" }
else if methodName = "getColor" then
  { varname = "getColor"; vartype = "int" }
else if methodName = "flipImageHorizontal" then
  { varname = "flipImageHorizontal"; vartype = "void" }
else if methodName = "flipImageVertical" then
  { varname = "flipImageVertical"; vartype = "void" }
else if methodName = "saveImage" then
  { varname = "saveImage"; vartype = "void" }
else if methodName = "meanFilter" then
  { varname = "meanFilter"; vartype = "void" }
else if methodName = "rotate" then
  { varname = "rotate"; vartype = "void" }
else if methodName = "compare" then
  { varname = "compare"; vartype = "boolean" }
else if methodName = "sharpen" then
  { varname = "sharpen"; vartype = "void" }
else if methodName = "edgeDetection" then
let javacodespit = "
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import java.awt.image.BufferedImage;
import java.awt.image.ConvolveOp;
import java.awt.image.Kernel;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

class showImage extends JPanel
{
    BufferedImage x;
    Dimension panelDimension;

    public showImage(BufferedImage xin)
    {
        x = xin;
        panelDimension = new Dimension(xin.getWidth(), xin.getHeight());
    }

    @Override
    protected void paintComponent(Graphics graphics)
    {
        super.paintComponent(graphics);
        int width = (getWidth() - panelDimension.width)/2;
        int height = (getHeight() - panelDimension.height)/2;
        graphics.drawImage(x, width, height, this);
    }
}
public class yail {

    static BufferedImage newImage(String image) {
        BufferedImage xin = new BufferedImage(800, 800, BufferedImage.TYPE_3BYTE_BGR);
        try {
            xin = ImageIO.read(new File(image));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return xin;
    }

    static void printImage(BufferedImage xin) {
        try {
            showImage x = new showImage(xin);
            x.setOpaque(true);
            JFrame f = new JFrame("Image");
            f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            f.setContentPane(x);
            f.setSize(800, 600);
            f.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
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 int getAlpha(int color)
{
    return (color >> 24) & 0xff;
}

public static int getRed(int color)
{
    return (color >> 16) & 0xff;
}

public static int getGreen(int color)
{
    return (color >> 8) & 0xff;
}

public static int getBlue(int color)
{

return(color) & 0xff;

public static int getColor(BufferedImage im, int x, int y)
{
    return im.getRGB(x, y);
}

public static BufferedImage flipImageHorizontal(BufferedImage xin)
{
    AffineTransform tx = AffineTransform.getScaleInstance(1, -1);
    tx = AffineTransform.getScaleInstance(-1, 1);
    tx.translate(-xin.getWidth(null), 0);
    AffineTransformOp op = new AffineTransformOp(tx,
        AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
    op = new AffineTransformOp(tx, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
    BufferedImage xout = op.filter(xin, null);
    return xout;
}

public static BufferedImage flipImageVertical(BufferedImage xin)
{
    AffineTransform tx = AffineTransform.getScaleInstance(1, -1);
    tx.translate(0, -xin.getHeight(null));
AffineTransformOp op = new AffineTransformOp(tx, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
BufferedImage xout = op.filter(xin, null);
return xout;
}

public static void saveImage(BufferedImage xin,String imagePathName)
{
    File file = new File(imagePathName + ".jpg");
    try {
        ImageIO.write(xin, "jpg", file);  // ignore returned boolean
    } catch(IOException e) {
    }
}

public static BufferedImage meanFilter(BufferedImage im)
{
    float[] elements = { 1/9f, 1/9f, 1/9f,
                        1/9f, 1/9f, 1/9f,
                        1/9f, 1/9f, 1/9f};

    BufferedImage xout = new BufferedImage(800, 800, BufferedImage.TYPE_3BYTE_BGR);
    Kernel kernel = new Kernel(3, 3, elements);
    ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP,null);
    cop.filter(im,xout);
    return xout;
public static BufferedImage rotate(float theta, BufferedImage xin)
{
    AffineTransform tx = AffineTransform.getScaleInstance(1, -1);
    tx = AffineTransform.getScaleInstance(-1, 1);
    tx.rotate(12312.33 * Math.PI / 180);
    AffineTransformOp op = new AffineTransformOp(tx,
        AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
    op = new AffineTransformOp(tx, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
    BufferedImage xout = op.filter(xin, null);
    return xout;
}

public static Boolean compare(BufferedImage xin1, BufferedImage xin2)
{
    int flag = 0;
    for (int i=0; i < xin1.getHeight() && flag == 0; i++)
    {
        for (int j=0; j < xin1.getWidth() && flag == 0; j++)
        {
            if(xin1.getRGB(j, i) != xin2.getRGB(j, i))
            {
                flag = 1;
            }
        }
    }
    return flag == 0;
}
if(flag==1)
{
    return false;
}
else
{
    return true;
}

public static BufferedImage sharpen(BufferedImage im)
{

    float[] elements = {
        -1f, -1f, -1f,
        -1f, 12f, -1f,
        -1f, -1f, -1f
    };

    BufferedImage xout = new BufferedImage(800, 800, BufferedImage.TYPE_3BYTE_BGR);
    Kernel kernel = new Kernel(3, 3, elements);
    ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
    cop.filter(im, xout);
    return xout;
}

public static BufferedImage edgeDetection(BufferedImage im)
{

}
float[] elements = { 0.0f, -1.0f, 0.0f,
                    -1.0f, 4.f, -1.0f,
                    0.0f, -1.0f, 0.0f};

BufferedImage xout = new BufferedImage(800, 800, BufferedImage.TYPE_3BYTE_BGR);
Kernel kernel = new Kernel(3, 3, elements);
ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
cop.filter(im, xout);
return xout;

public static BufferedImage getBuff(String image)
{
    BufferedImage xin = new BufferedImage(800, 800, BufferedImage.TYPE_3BYTE_BGR);
    try {
        xin = ImageIO.read(new File(image));
    } catch (IOException e) {
        e.printStackTrace();
    }
    return xin;
}

public static void deleteImage(String imagePathName)
{
    (new File(imagePathName)).delete();
}
public static BufferedImage putColor(BufferedImage imageBuff, int color, int x, int y)
{
    imageBuff.setRGB(x, y, color);
    return imageBuff;
}

public static int makeColor(int alpha, int red, int green, int blue)
{
    return (alpha << 24 | red << 16 | green << 8 | blue);
}

public static int getWidth(BufferedImage img)
{
    return img.getWidth();
}

public static int getHeight(BufferedImage img)
{
    return img.getHeight();
}

static Kernel filterinit()
{
    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
    });
}

static float getFilterData(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 setFilterData(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;
}


let javaMethodDeclare javaMethod =
if javaMethod.fname = "main" then "public static void main(String[] args)"
else
    let string1 = List.fold_left (fun f_list formal -> (javaParameterType formal.vartype ^ " " ^ formal.varname):f_list) [] javaMethod.formals
    in
"public static " ^ javaParameterType javaMethod.rettype ^ " " ^ javaMethod.fname ^ "(" ^ String.concat ","
(List.rev string1) ^ ")"

let rec numberOfElements (name, num, l) =
  if num = 0 then l
  else numberOfElements (name, num-1, name::l)

let rec symbolSearch symbols identifier =
  let checklocalSymbol = List.exists (fun temp -> if temp.varname = identifier then true else false) symbols
  in
  if checklocalSymbol then
    let variable = List.find (fun temp -> if temp.varname = identifier then true else false) symbols
    in
    variable
  else
    let checkGlobalSymbol = List.exists (fun temp -> if temp.varname = identifier ^ "_global_var" then true else false) symbols
    in
    if checkGlobalSymbol then
      let variable = List.find (fun temp -> if temp.varname = identifier ^ "_global_var" then true else false) symbols
      in
      { varname = identifier; vartype = variable.vartype }
    else
      raise (Failure ("Error 480 : JavaGen unrecognized symbol"))
  else
    raise (Failure ("Error 480 : JavaGen unrecognized symbol"))

let rec javaGen symbols = function
  IntLiteral(x) -> string_of_int x
| FloatLiteral(x) -> "(float)" ^ string_of_float x
| StringLiteral(x) -> "\"" ^ x ^ \\
}
| Bool(x) -> x
| Uminus(x) -> "-(" ^ javaGen symbols x ^ ")"
| Not(x) -> "!(" ^ javaGen symbols x ^ ")"
| Id(x) -> x
| Paren(x) -> "(" ^ javaGen symbols x ^ ")"

| Binop(leftChild, o, rightChild) ->

  let leftChild_bk = expressionClassification symbols leftChild
  in
  let rightChild_bk = expressionClassification symbols rightChild
  in
  if leftChild_bk.vartype = "image" then
    let op = (match o with
      Convolve -> "Convolve"
      | Equal
      | Neq
      | Less
      | Leq
      | Greater
      | Geq
      | Add
      | Sub
      | Mult
      | Div
      | And
      | Or ->

      raise (Failure ("Error 505: JavaGen - Yail allows only # operation on images "))
    )
  else
    raise (Failure ("Error 505: JavaGen - Yail allows only # operation on images "))
  )

in

if op = "Convolve" then
  if rightChild_bk.vartype = "filter" then
    "convolve(" ^ javaGen symbols leftChild ^ ", " ^ javaGen symbols rightChild ^ ")"
  else
    raise (Failure ("The second operand of '#' must be a filter type"))
  else
    raise (Failure ("Error 514: JavaGen - Yail allows only # operation on images "))
else
  javaGen symbols leftChild ^ " " ^ 
    (match o with
      Add -> "+"
      | Sub -> "-"
      | Mult -> "*"
      | Div -> "/
      | Equal -> "=="
      | Less -> "<"
      | Leq -> "<="
      | Greater -> ">
      | Geq -> ">="
      | And -> "&&"
      | Neq -> "!="
      | Or -> "| |
      | Convolve -> raise (Failure ("Error 533: JavaGen - Yail allows # operation only on images ")))
  ^ " " ^

  javaGen symbols rightChild
| Assign(x1, x2) ->
| x1 ^ " = " ^ javaGen symbols x2

| Call(name, elements) ->
| let symbols_list = numberOfElements (symbols, List.length elements, [])
in
| yailMethodConversion name ^ "(" ^ String.concat ", " ^ (List.map2 javaGen symbols_list elements) ^ ")"

| Datatype(_) -> "" (* Print parse error *)

| NewFilter(x1, x2) ->
| "kernelnew(" ^ javaGen symbols x1 ^ ", " ^ javaGen symbols x2 ^ ")"

| FilterAccess(filter, x1, x2) ->
| "getFilterData(" ^ filter ^ ", " ^ javaGen symbols x1 ^ ", " ^ javaGen symbols x2 ^ ")"

| AssignFilter(identifier, x1, x2, floatvalue) ->
| identifier ^ " = setFilterData(" ^ identifier ^ ", " ^ javaGen symbols x1 ^ ", " ^ javaGen symbols x2 ^ " ", " ^ javaGen symbols floatvalue ^ ")"

let rec valueconvert arry lst =
  let oneD = List.rev lst
  in arry @ oneD

let rec codeGen symbols = function
  Block(stmts) ->
  let symbols_list = numberOfElements (symbols, List.length stmts, [])
in
Expr(expr) -> begin
match expr with
  | IntLiteral(_) -> "";
  | FloatLiteral(_) -> "";
  | StringLiteral(_) -> "";
  | Id(_) -> "";
  | Uminus(_) -> ""
  | Binop(_, _, _) -> ""
  | _ -> javaGen symbols expr ^ "\n";
end

If(expression, statement, Block([])) -> "if (^ javaGen symbols expression ^ "\n")^ codeGen symbols statement

If(expression, statement1, statement2) -> "if (^ javaGen symbols expression ^ "\n")^ codeGen symbols statement1 ^ "else\n"^ codeGen symbols statement2

For(expression1, expression2, expression3, statement) -> "for (^ javaGen symbols expression1 ^ ";"^ javaGen symbols expression2 ^ ";"^ javaGen symbols expression3 ^ "')^ "\n"^ \n"^ codeGen symbols statement ^ "\n"^ \n"

FilterInit(filter, lst) ->
  let temp1 = List.fold_left valueconvert [] lst in
  let temp2 = numberOfElements (symbols, List.length temp1, []) in
  let temp3 = List.map2 javaGen temp2 temp1 in
  let a = string_of_int (List.hd (List.map (fun temp4 -> List.length temp4) lst)) in
  let b = string_of_int (List.length lst)
in

    filter ^ " = new Kernel (\^ a ^", ^ b ^", new float[]{\^ (String.concat "", temp3) ^"});

let retvar_return identifier =
    if(identifier.fname = "main") then ""
else if(identifier.rettype = "Error") then raise (Failure ("Cannot return a literal"))
else if (identifier.rettype = "") then ""
else "return ^identifier.retname^";

let declGen identifier = javaParameterType identifier.vartype ^ " " ^ identifier.varname ^ " = " ^
javaParameterIntialize identifier.vartype ^ ";\n"

let returnGen identifier =
    if (identifier.rettype = "" || identifier.fname = "main") then ""
else ""

let functionGen identifier globals =
    let symbols = globals @ identifier.locals @ identifier.formals @ [{} varname = "" ; vartype = identifier.rettype]
    in
    let temp1 = numberOfElements (symbols, List.length identifier.body, [])
    in
    javaMethodDeclare identifier ^ "\n{" ^
returnGen identifier ^
    String.concat "" (List.map declGen identifier.locals)^
String.concat "" (List.map2 codeGen temp1 identifier.body)^
retvar_return identifier ^
"
};\n"
let globalgen identifier =

"public static " ^ javaParameterType identifier.vartype ^ " " ^ identifier.varname ^ " = " ^
javaParameterInitialize identifier.vartype ^ ";\n"

let programGen (variables, methods) =

let global_methods = List.map (fun a -> { varname = a.fname ^ "_global_func"; vartype = a.rettype }) methods in

let global_variables = List.map (fun a -> { varname = a.varname ^ "_global_var"; vartype = a.vartype }) variables in

let globals = global_methods @ global_variables (* concatenate global variables and global functions to form symbols *) in

let globals_list = numberOfElements (globals, List.length methods, []) in

javacodespit ^
String.concat "\n" (List.map globalgen variables) ^ "\n" ^
String.concat "\n" (List.map2 functionGen methods globals_list) ^ "\n\n" ^
"\n"