SIPL: Simple Image Processing Language

Shanshan Zhang, Yihan Zhao, Yuedong Wang, Ci Chen, Simon Zhai
{sz2648, yz2996, yw2931,cc4192, yz3116}@columbia.edu

Columbia University
COMS W4115: Programming Languages and Translators

December 20, 2017

Contents

Chapter 1

Introduction

Background

Simple Image Processing Language is a language specifically targeted for image processing. We intend on implementing a language that can deal with images in an effective and fast way. It simplifies the implementation of image processing algorithm compared to using languages like C, C++ by converting image into a Image, which is represented by channels, each of which is a primitive data type - Mat. Mat is a 2d matrix and all other operations are based on this data type, which makes SIPL simple and compact.

SIPL not only supports basic calculation for images, but also supports more advanced image manipulations. For example, users could ask to resize, flip and rotate the images. Also, our language allow users to change an image into grey, detect the edge of an image and separate the image into three-primary colors.

Features

SIPL supports auto Clamping, auto in-bound and some auto type conversion features. The language simplifies the programming of image Processing.

Chapter 2

Language Tutorial

2.1 Compile and Run Your Program

1. Compile SIPL compiler.

   Inside SIPL folder, type “make” in the terminal. It will produce compiler files for our compiler, including compiler file *sipl.native* and library files like *lib/utils.o*.

2. Testing

   To run all the test, type the following command in the command line: “./testall.sh”

   To run certain test, type the following command in the command line:

   “./testall.sh test-file-name.sp”

3. Run a SIPL program.

   SIPL program filename ends with “.sp”.

   To compile the program, type the following command in the command line: “./SIPLcompiler.sh test-program.sp”. It will compile the source code, link with our library, and produce the executable file “test-program.exe”.

   To run the program, type “./test-program.exe” in the command line.

2.2 Basic Types

- **Image**

Image img;	declare an Image variable *img*
var = img[i][j][“g”];	access “g” (green) channel at *i* row *j* column of *img*
img[i][j][“r”] = val;	assign integer *val* to “r” red channel at [i][j] position of *img*
img.width	get width of image
img.height	get height of image

- **Matrix**

Matrix mat;	declare a Matrix variable *mat*
mat = [0.1, 0.2; 0.3, 0.4; 0.5, 0.6];	initialize a matrix
var = mat[i][j];	access value at *i* row *j* column of matrix *mat*
mat[i][j] = var;	assign float value var to matrix at [i][j] position
mat.width	get width of matrix
mat.height	get height of matrix

- Pixel

Pixel pix;	declare a Pixel variable *pix*
pixel = img[i][j][“a”]	access Pixel (All red, green, blue channels)
img[i][j][“a”] = pix;	assign Pixel to img

- Type conversion

int -> float	int2float()	SIPL can do auto-casting
float -> int	float2int()	require manual cast

2.3 Full example

In this section, you will see two full program examples that are written in SIPL. You could see the power of our language processing language. The code is very concise and easy to understand.

2.3.1 Add Two Images

In that example, we tried to add two images together. You can follow the below steps. The input and output are shown in the pictures below.

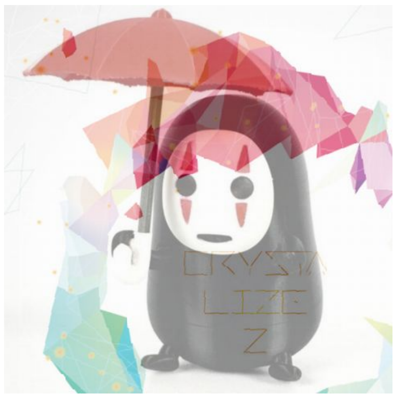
Read the two images;

1. Do gaussian regression on the 1st image ;
2. Rescale the 2nd image base on the first image’s scale;
3. Do add operation on two images;
4. Write the output.

```
int main(){

    float scalex;
    float scaley;
    Matrix mat;
    Matrix kern;
    Image img;
    Image img2;
    Image img3;
    Image img4;

    img = imgread("10.jpg");
    img2 = imgread("9.jpg");


    scalex = int2float(img.width) / img2.width;
    scaley = int2float(img.height) / img2.height;

    kern = gaussian(3, 0.5);
    img = img ** kern;


    img2 = resize(img2, 0.95, 0.95);
    img2 = sliceImg(img2, 0, 0 ,img.height, img.width );
    img2 = img * 0.45 + img2 * 0.55;

    imgwrite(img2, 0, "overlapg.png");

}
```

9.jpg

10.jpg

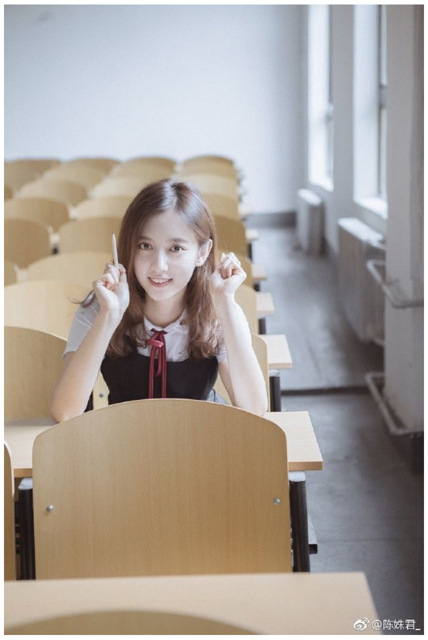

overlapg.png

2.3.2 Gaussian regression

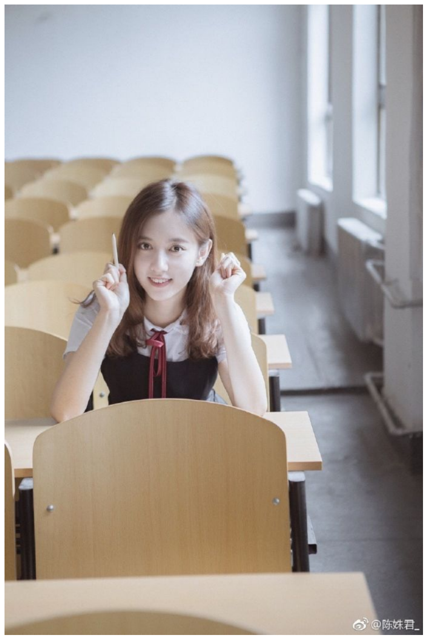
In that section we tried to do gaussian regression with different level and alpha value on the same input image. The input and output are shown in the pictures below.

1. Read the image;
2. Create different gaussian kernels;
3. Do convolution will different kernels on the image;
4. Write the output.

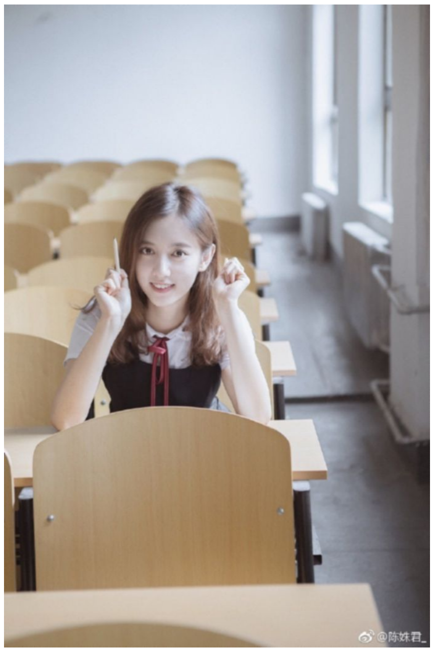
```
int main(){

    Matrix kern;
    Image img;
    Image img2;

    img = imgread("3.jpg");


    kern = gaussian(3, 0.5);
    img2 = img ** kern;

    imgwrite(img, 0, "gauss1.png");

    kern = gaussian(3, 1.0);
    img2 = img ** kern;

    imgwrite(img, 0, "gauss2.png");

    kern = gaussian(5, 1.0);
    img2 = img ** kern;

    imgwrite(img, 0, "gauss3.png");

}
```



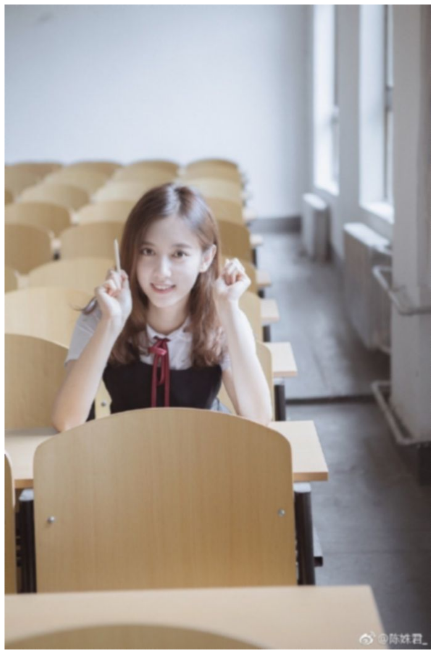
3.jpg



gauss1.png



gauss2.png



gauss3.png

Chapter 3

Language Reference Manual

The SIPL language is a linear algebra manipulation language specially targeted for image processing. It uses an efficient way to express complex image manipulation algorithms to complete matrix operations. Features defining in the language include basic image operations like gray conversion, image rotation, image filtering and resizing. This manual describes in detail the lexical conventions, identifiers, expressions, operators, statements and built-in functions of the SIPL language.

3.1 Lexical Conventions

3.1.1 Tokens

There are five kinds of tokens in SIPL: Keywords, identifiers, constants, control characters and operators. Blanks, tabs, newlines and comments are ignored except that they serve to separate tokens.

3.1.2 Comments

The characters /* introduces a comments which terminates with */.

/*This is a sample comments*/

3.1.3 Keywords

The reserved keywords in SIPL are:
if
else
for
while
return
int
float
bool
Image
Matrix

Pixel
string
void
true
false
width
height

3.1.4 Identifiers

Identifiers are composed of a lower or upper-case letter immediately followed by any number of additional letters and/or digits. Identifiers are case sensitive, “Simon” and “simon” are different identifiers. Keywords and underscores are not allowed to be identifiers.

3.1.5 Constants

Numeric constants :
Integers are represented by a series of number characters. Floats are represented by a series of number following a decimal point and a series of numbers.

3. 2 Meaning of Identifiers

3.2.1 Data Types
3.2.1.1 Basic Type

Int: 32 bit integer
Example:
```
int var;
var = 1;
```

Float: 32 bit real number
Example:
```
float var;
var = 1.0;
```

Boolean: A constants with either True or False
Example:
```
bool b;
b = true;
```

```
b = false;
```

3.2.1.2 Built-in type

Matrix
n×m matrix of float values.
Example:

```
Matrix mat;               // declare an variable of matrix type.
mat = [0.1, 0.2; 0.3, 0.4];     // matrix initialization.
var = mat[i][j];          // access value at i row j column of matrix mat
mat[i][j] = var;          // assign float value var to matrix at [i][j] position
mat.width                 // get width of matrix
mat.height                // get height of matrix
```

Image
n×m×3 size of Unit8 3 dimensional matrix. SIPL provides **auto-clamping** for Image data type. When assigning values to image, they are constrained within (0 - 255) automatically.
Example:

```
Image img;
img = imgCreate(100, 100);   // create a 100 by 100 empty image
intVar = img[i][j][“g”];     //access “g” (green) channel at i row j column of img
img[i][j][“r”] = val;        // assign integer val to “r” red channel at [i][j] position of img
img.width                    // get width of image
img.height                   // get height of image
img[i][j][“r”] = 300;        // 300 will be constrained to 255 when assigning to an image.
```

Pixel
Unit8 vector with 3 elements. Each element represent the value of one of the RGB channel.
Example:

```
Image img;
Pixel p;
img = imread(“1.jpg”);     //get an image
p = img[50, 70, “a”];      // get the specific pixel from image
```

3.3 Type Conversion

SIPL provides auto type conversion for integer to float, and *float2int()* built-in function for float type to integer type.
Example:

```
float fvar;
```

```
int i;
fvar = 1;               //auto integer to float conversion
i = float2int(2.5);     // use built-in function to convert float to integer.
fvar = 2 * 1.5;          // integer value 2 is auto converted to float value.
```

3.4 Operators

3.4.1 Basic Operators

3.4.1.1 +,-,*,/

These for operators have the same function in C, we can use these four operators to our Unit8, int, and float type data.

3.4.2 Image Operators

+, -, * is overloaded and used as image operators. Operations for image will result in changes in original image, because creating a new image each time for an operation is not efficient and will lose pointer for that intermediate images.

3.4.2.1 img + img

Add the pixel value of two images at the same position and return a new image, this addition can be used to mix 2 pictures.

Example:

```
Image img1;
Image img2;
Image img3;

img1 = imgread("1.jpg");
img2 = imgread("2.jpg");
Img3 = img1 + img2;
```

3.4.2.2 img + int

Add a fix number to all pixels in one image and return a new image, this addition can be used to increase the brightness of the original picture.

Example:

```
Image img1;
Image img2;

img2 = img1 + 5; // img2 will be brighter than img1
```

3.4.2.3 img - int

Subtract a fix number to all pixels in one image and return a new image, this subtract can be used to decrease the brightness of the original picture.
Example:

```
Image img1;
Image img2;

img2 = img1 - 5; // img2 will be darker than img2
```

3.4.2.4 img * float

Multiply a float number to all pixels in one image and return a new image, this multiplication can be used to increase/decrease the brightness of a image(set the multiplier >1 will increase brightness <1 will decrease brightness).
Example:

```
Image img1;
Image img2;

img2 = img1 * 0.5;   //img2 will be half brightness as image1
```

3.4.3 Pixel Operators

Use between int, float and Pixel. It will group from left to right, ”*” have higher priority than ”+” and ”-”.

3.4.3.1 pixel + pixel

Add the vector value of two pixels together and return a new pixel.
Example:

```
Image img;
Pixel p1;
Pixel p2;
Pixel p3;
img = imread(“1.jpg”);     //get an image
p1 = img[50, 70, “a”];    // get the specific pixel from image
p2 = img[50, 69, “a”];
p3 = p1 + p2;
```

3.4.3.2 pixel + int

Add the same value to all channels of a pixel and return a new pixel.
Example:

```
Image img;
pixel p1;
pixel p2;
```

```
img = imread("1.jpg");     //get an image
p1 = img[50, 70, "a"];    // get the specific pixel from image
p2 = p1 + 3;    // p2 will be brighter than p1
```

3.4.3.3 pixel - int

Subtract the same value from all channel of a pixel and return a new pixel.

Example:

```
Image img;
pixel p1;
pixel p2;

img = imread("1.jpg");     //get an image
p1 = img[50, 70, "a"];    // get the specific pixel from image
p2 = p1 - 3;    // p2 will be darker than p1
```

3.4.3.4 pixel * float

Multiply a number to all channels of a pixel.

```
Image img;
pixel p1;
pixel p2;

img = imread("1.jpg");     //get an image
p1 = img[50, 70, "a"];    // get the specific pixel from image
p2 = p1 * 0.5;    // p2 will have half brightness of p1
```

3.4.4 Convolution

Image ** Matrix

Matrix ** Image

Apply convolution to image with the given matrix as kernel. The operation will result in changes in the original image. Namely, SIPL will not create new Image for each convolution operation.

Example:

```
Image img;
Matrix mat;
mat = kernel("gaussian");  // create a kernel matrix
img = img ** mat;   // do convolution for img and matrix.
img = mat ** img;
img = img ** mat ** mat;  // ** operators can be applied multiple times in an expression
```

3.6 Statements
3.6.1 Loop Statements
3.6.1.1 "While" statement
“While” will execute a block that defined by user, if a specific expression is True:

```
while(expr){
        Statement
}
```

3.6.1.2 "For" statement
The for statement has the form:

```
for (expression1;expression2;expression3) statement
```

Which is equal to:

```
expression-1;
while ( expression-2 ) {
statement
expression-3;
}
```

3.7 Build-in Functions

3.7.1 Image Creation
img = imgCreate(height, width)
Create an image with given size.

3.7.2 Image Input
img = imgread(“myPicture.png”)
Read an image.

3.7.3 Image Copy
img = imgCopy(img)
Copy an image from the original image.

3.7.4 Image Output
imgwrite(img, “picture.png”)

Output an image.

3.7.5 Gray Conversion
img = changeGrey(img)
Change colored image into gray image.

3.7.6 Image Rotation
rotatedImg = rotateImg(img)
Clockwise rotate an image by pi/2.

3.7.7 Image Flipping
flippedImg = flipImg(img, n)
Flip an image(n=1 for horizontal flip and n=-1 for vertical flip, otherwise cast an error).

3.7.8 Matrix Construction
Use Matrix to construct a matrix.

3.8 SIPL Library Functions (written in SIPL)

3.8.1 Image Resizing
img = resize(img, scaleX, scaleY)
Resize the image by multiplying two numbers to its width and height.

3.8.2 Image/Matrix cropping
img = sliceImg(img, left, up, width, height)
mat = sliceMat(mat, left, up, width, height)
Crop an image/matrix, from the given position (left, up) with given size (width, height).

3.8.3 Gaussian Kernel Construction
mat = gaussian(size, sigma)
Create a Gaussian matrix with given size and given variance.

3.9 Sample Code

3.9.1 Copy an Image

```
Image img;
Image imgC;
```

```
/* Initialization of tow images */
img = imgread(“lena.png”);
/* Read an image */
imgC = imgCopy(img);
/* Copy an image */
imgwrite(imgC, 0, “lena2.png”);
/* Write an image */
```

3.9.2 Sharpen an Image

```
Image img;
Image imgS;
Matrix mat;
/* Initialization two images and an kernel */
img = imgread(“lena.png”);
/* Read an image */
mat = kernel(“sharpen”);
/* Automatically generate a sharpen kernel */
imgS = img ** mat;
/* Apply  convolution */
imgwrite(imgS, 0, “lena2.png”);
/* Write an image */
```

3.9.3 Edge Detection of an Image

```
Image img;
Image imgE;
Matrix mat;
/* Initialization two images and an kernel */
img = imgread(“lena.png”);
/* Read an image */
mat = kernel(“edge”);
/* Automatically generate a edge detection kernel */
img = img ** mat;
/* Apply  convolution */
imgwrite(imgS, 0, “lena2.png”);
/* Write an image */
```

Chapter 4

Project Planning

4.1 Project Management Overview

There should be a particular order of project management processes to finish the whole project. The project management process of SlPL follows the following steps: Initiating (Start), Planning(Plan), Execution(Do), Monitoring (Check), Closing (End).

4.1.1 Initiating (Start)

At this stage, we assigned our team for the SIPL project. Each team member had his unique role in the team and sponsored part of the project.

4.1.2 Planning(Plan)

At this stage we completed end-to-end project planning. We prioritized the project, calculated the cost and made a schedule, and determine what resources are needed.

4.1.3 Execution(Do)

We adapted an iterative planning during working process. Initially, we set the main goals and important deadlines for building the SIPL compiler. At the same time, we iteratively created short-term goals as we go deeper.

4.1.4 Monitoring (Check)

At that stage, we generated the unit test for our compiler and made proper adjustment and improvement of our SIPL.

4.1.5 Closing(End)

This is the final stage of the project. We collected all the resources and gathered all our work to accomplish the final report and presentation ppt and got prepared for the project demo.

4.2 Specification, Development and Testing

Based on the deadlines of the project deliverables were set, we had an initial idea about our project milestones should look like. After we completed the language proposal and the language reference manual, we had a more comprehensive understanding of the scope of our project. Then, we implemented necessary features and specifications for our language.

The Development process followed the stages of the compiler architecture, starting with the scanner, followed by the parser, semantics checker and code generator. Each of our feature was implemented followed by those process.

Before we finally completed, we implemented unit tested for each function. We designed tests for various features in our language in order to check how they pass through the entire compiler and output the desirable answer.

4.3 Team Responsibilities

Following is the table for team responsibilities.

Team Member	Responsibility
Yuedong Wang	System Architect, Language Guru
Ci Chen	Assistant System Architect, Language Guru
Yuexiang Zhai	Language Guru
Yihan Zhao	Tester
Shanshan Zhang	Project Manager

4.4 Project Timeline

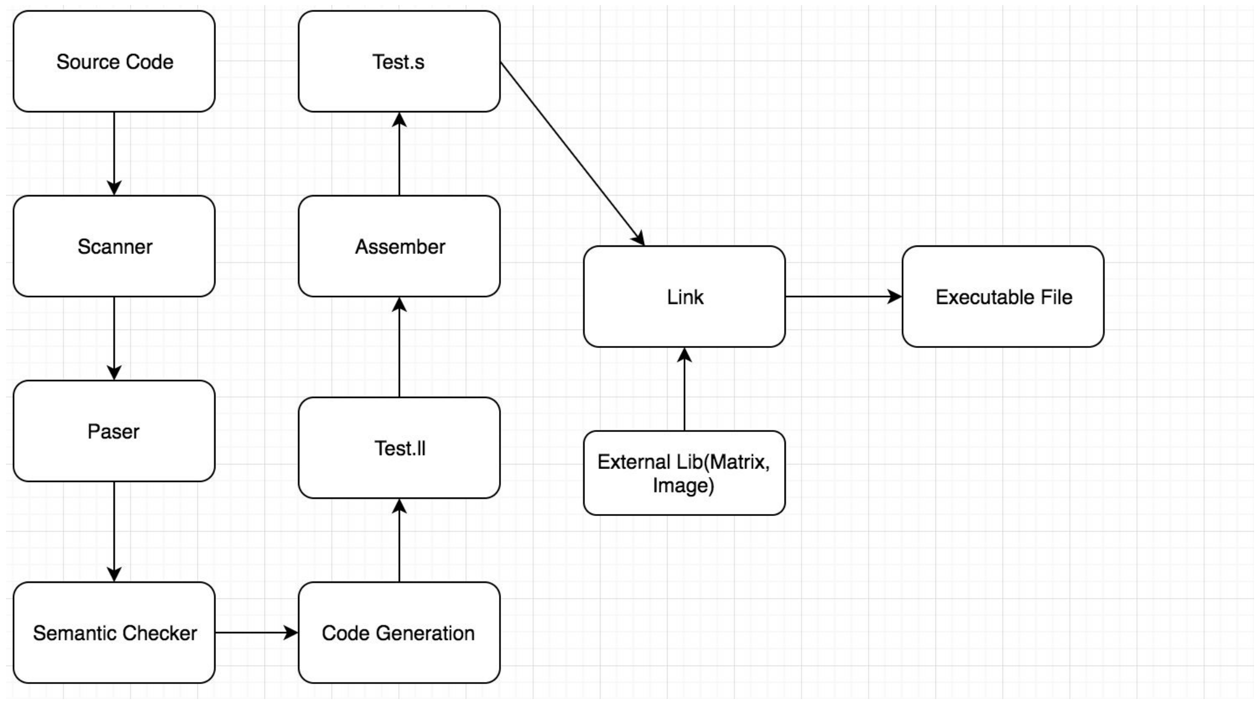
Date	Milestone
September 26	Complete Language proposal
October 16	Complete Language Reference Manual
October 31	Compiler front end (lexer and parser) complete

November 16	Semantics / type checking complete
November 25	Code generation complete
December 6	Hello World runs
December 17	Finalize the test suite and bug fix
December 19	Demo
December 20	Final report complete

Chapter 5

Architectural Design

5.1 diagram



5.2 Scanner

The scanner takes source code and converts it to lists of tokens. Comments as well as whitespace are removed from the tokens.

5.3 Parser and AST

The parser takes tokens generated from the scanner as the input and constructs an abstract syntax tree based on our SIPL grammar. The AST tree represents the overall structure of our program.

5.4 Semantic Checker

The semantic checker takes The abstract syntax tree generated by the parser as the input. Then, the AST passes through the list of global statements and check the semantic errors. The exception is raised while errors occurs.

5.5 Code Generator

The code generator traverses the abstract syntax tree then producing LLVM IR. Those building process will go through the statements and expressions iteratively.

5.6 External libraries

For external libraries, we wrote it in C. Our library includes Matrix and Image.

Chapter 6

Test

6.1 Syntax Verification

Syntax verification aims at confirming that the parser accepts all valid strings and rejects all invalid ones defined in LRM. Several examples are shown as follow.

- Matrix syntax verification

fail-syntax0.sp defined a matrix without the closing brackets, it should not compile.

```
int main () {
   Matrix mat;
   mat = [1.0, 1.0; 1.0, 1.0;
}
```

fail-syntax1.sp defined a matrix with extra bracket, according to the LRM it should not compile.

```
int main () {
   Matrix mat;
   mat = [1.0, 1.0;] 1.0, 1.0];
}
```

- Nested assignment expression

test-sytax1.sp tried to make a nested assignment and write the two images, it should compile.

```
int main () {
   Image a;
   Image b;
   Image c = imgread("lena.png");
   a = b = c;
   imgwrite(a, 0, "sytnax-testa.png");
   imgwrite(b, 0, "sytnax-testb.png");
}
```

}

6.2 Semantic Verification

Semantic verification is used to confirm that the verifier accepts all valid parse trees and rejects all invalid ones. Several semantic test examples are shown as follow.

- Image Read Semantic Check

test-imgread.sp tried to read an image file and write it as an output file, it should compile.

```
int main () {
        Image img;
        img = imgread("lena.png");
        imgwrite(img, 0, "lena33.png");
}
```

- Matrix Access Semantic Check

test-matrix-access.sp defined a matrix and tried to have access to a certain index, it should compile.

```
int main () {
        Matrix mat;
        int i;
        i = 2;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        printfloat(mat[1][i]);
}
```

The compile result is shown in test-matrix-access.out.

```
6.000
```

fail-matrix-access1.sp defined a matrix and tried to have access to a certain index with invalid type, it should not compile.

```
int main () {
        Matrix mat;
        mat = [0, 0, 0; 0, 0, 0; 0, 0, 0];
        printfloat(mat[1][2.0]);
}
```

The error message is shown in fail-matrix-access1.err.

```
Fatal error: exception Failure("specified channel is not valid")
```

fail-matrix-access2.sp tried to have access to a index of a int, it should not compile.

```
int main () {
        Matrix mat;
        int fakeMat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        printfloat(fakeMat[1][2]);
}
```

The error message is shown in fail-matrix-access2.err

```
Fatal error: exception Failure("identifier is not matrix")
```

- Matrix Assign Semantic Check

test-matrix-assign0.sp tried to assign a matrix and print it, it should compile.

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        printMatrix(mat);
}
```

The compile result is shown in test-matrix-assign0.out.

```
3
3
1.0000002.0000003.000000
4.0000005.0000006.000000
7.0000008.0000009.000000
```

test-matrix-assign.sp tries to assign 12.0 to the certain index of the matrix and print it, it should compile.

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        mat[1][2] = 12.0;
        printMatrix(mat);
}
```

The compile result is shown in test-matrix-assign.out.

```
3
3
1.0000002.0000003.000000
4.0000005.00000012.000000
7.0000008.0000009.000000
```

fail-matrix-assign1.sp tried to assign to an invalid index, it should not compile.

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        mat[1][2.0] = 12.0;
        printMatrix(mat);
}
```

The error message is shown in fail-matrix-assign1.err.

```
Fatal error: exception Failure("Matrix index should be Integer")
```

fail-matrix-assign2.sp tried to assign to the invalid identifier, it should not compile.

```
int main () {
        Matrix mat;
        int fakeMat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        fakeMat[1][2] = 12.0;
        printMatrix(mat);
}
```

The error message is shown in fail-matrix-assign2.err.

```
Fatal error: exception Failure("identifier is not matrix")
```

- Test String

test-string.sp tried to test the string of SIPL, it tried to print string in two different ways, it should compile.

```
int main()
{
  string str;
  str = "test string";
  prints("test string");
  prints(str);
  return 0;
}
```

The compile result is shown in test-string.out.

```
test string
test string
```

6.3 Image Processing Verification

In that section, we generated both simple and complicated programs that test all the functions of the language such as read and write image, image manipulations. Some examples are shown as follow.

- Test Image Add

test-img-imgAdd.sp tried to read an image and add two images, it should compile.

```
int main () {
        Image img;
        img = imgread("lena.png");
        img = img + img;
        imgwrite(img, 0, "lena-imgAdd.png");
}
```

- Test Image Copy

test-img-imgCopy.sp tried to read an image and then copy the image, it should compile.

```
int main () {
        Image img;
        Image img2;
        img = imgread("lena.png");
        img2 = imgCopy(img);
        imgwrite(img2, 0, "lena-imgCopy.png");
}
```

- Test Image Creation

test-img-imgCreate.sp tried to create an new image and write it as an image, it should compile.

```
int main () {
        Image img;
        img = imgCreate(100, 100);
        imgwrite(img, 0, "lena-imgCreate.png");
}
```

- Test Image Rotation

test-img-rotateImage.sp tried to read an image, rotate it, write it as an image, it should compile.

```
int main () {
        Image img;
        img = imgread("lena.png");
        rotateImage(img);
        imgwrite(img, 0, "lena-rotate.png");
}
```

- SIPL Library Test

sipllib.sp tried to test the functions that are implemented in SIPL such as: resize(), gaussian(), sliceImg(), sliceMat().

```
int main () {
Image resize (Image img, float scaleX, float scaleY){
   float cols;
   float rows;
   float bp_row;
   float bp_col;
   int i;
   int j;
   float delx;
   float dely;
```

```
    Pixel tmp;
    int tmp2;
    int indexy;
    int indexx;
    Image res;

    cols =  img.width * scaleX;
    rows =  img.height * scaleY;

    res = imgCreate(float2int(rows), float2int(cols));

    for(i = 0; i < float2int(rows); i = i + 1){
        for(j = 0; j < float2int(cols); j = j + 1){
            bp_row =  i / scaleY;
            bp_col =  j / scaleX;
            indexx = float2int(bp_col);
            indexy = float2int(bp_row);

            delx = bp_col -  indexx;
            dely = bp_row -  indexy;

            tmp =  img[indexy][indexx]["a"] * (1.0 - delx) * (1.0 - dely);
            tmp = tmp +  img[indexy + 1][indexx]["a"] * delx * ( 1.0 - dely) ;
            tmp = tmp +  img[indexy + 1][indexx + 1]["a"] * dely * delx;
            tmp = tmp +  img[indexy][indexx + 1]["a"] * (1.0 - delx) * dely;
            res[i][j]["a"] = tmp;

        }
    }

    return res;
}

Matrix gaussian(int size, float sigma){

    int i;
    int j;
    float e;
    float tmp;
    Matrix gauss;
    float sum;

    gauss = matCreate(size, size);
    e = 2.718281828459;
```

```
   for(i = 0; i < size; i = i + 1){
      for(j = 0; j < size; j = j + 1){
         tmp = 0 - (power((i - (size - 1)/2.0), 2.0) + power((j - (size - 1)/2.0), 2.0))/(2.0 * sigma
* sigma);
         gauss[i][j] = power(e, tmp);
         sum = sum + gauss[i][j];
      }
   }

   for(i = 0; i < size; i = i + 1){
      for(j = 0; j < size; j = j + 1){
         gauss[i][j] = gauss[i][j]/sum;
      }
   }
   return gauss;
}

Image sliceImg(Image img, int left, int up, int cols, int rows) {
   int i;
   int j;
   int down;
   int right;
   Image res;
   down = up + rows;
   right = left + cols;

   if(down >= img.height){
      down = img.height;
   }
   if(right >= img.width){
      right = img.width;
   }

   res = imgCreate(down - up, right - left);

   for(i = up; i < down; i = i + 1){
      for(j = left; j < right; j = j + 1){
         res[i - up][j - left]["r"] = img[i][j]["r"];
         res[i - up][j - left]["g"] = img[i][j]["g"];
         res[i - up][j - left]["b"] = img[i][j]["b"];
      }
   }
   return res;
}
```

```
Matrix sliceMat(Matrix mat, int left, int up, int cols, int rows) {
    int i;
    int j;
    int down;
    int right;
    Matrix res;
    down = up + rows;
    right = left + cols;

    if(down >= mat.height){
        down = mat.height;
    }
    if(right >= mat.width){
        right = mat.width;
    }

    res = matCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left] = mat[i][j];
        }
    }
    return res;
}
}
```

6.4 Test Automation

More than 80 unit tests have been written. To do the test automation, we wrote a script called testall.sh to compile and run all the test. It could recogize "tests/test-*.sp" and "tests/fail-*.sp" files. For tests that should success it will compile, run, and check the output and print Ok. For tests that should fail it will compile and check the error.

Chapter 7

Lessons Learned

- Yuedong: Yuedong: I learned the workflow of compiling and how to write a simple compiler. The most challenging part for me is actually to implement the first built-in function imgread(). That's when we figured out how to define a new type Image in C, define a pointer type in llvm and point to the actual type Image. And link with the built-in function written in C. Another challenging part is to add initialization for a Matrix variable. We need to figure out how to create a constant array and pass it to an external function to store the data. I enjoyed the time working with my teammates. Their attitude to contribute touched me a lot.

- Ci: How to work in a team.

- Simon: I learned how to work in team and trust my teammates.

- Yihan: I learned valuable skills in organizing a group project: conciseness can make both project management process and code elegant.

- Shanshan: Keep asking questions if you don't understand. Having a good team communication is very important, and Ci is very cute

Chapter 8

Appendix

8.1 lib - C

This folder contains the C library we wrote to support SIPL language

8.1.1 sipl.h

```
#ifndef SIPL_H
#define SIPL_H
/*
 * SIPL C Interface Header
 *
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include ...

struct img{
   int row;
   int col;
   unsigned char *r;
   unsigned char *g;
   unsigned char *b;
} ;


struct matrix{
         int row;
         int col;
         double *data;
} ;

struct  pixel {
         unsigned char r;
   unsigned char g;
   unsigned char b;
};
```

```
struct matrix* initMatrix(double * mat_data_ptr, int m, int n);
struct img*  imgread(const char*filename);
int imgwrite(struct img *mat, int fmt, const char *filename);
int changeGrey(struct img* image);
int rotateImage(struct img* image);
int flipImage(struct img* image, int n);
struct matrix* kernel(const char* str);
struct img* convImg(struct img *image, struct matrix *mat);
struct img* elementMult(struct img *img1, double num );
struct img* elementSub(struct img *img1, int num );
struct img* elementAdd(struct img *img1, int num );
struct img* imgAdd(struct img *img1, struct img *img2 );
struct img* imgCopy(struct img * img1);


#endif
```

8.1.2 stb_image.c

```
==========================    Contributors    ==========================

   Image formats                    Extensions, features
   Sean Barrett (jpeg, png, bmp)         Jetro Lauha (stbi_info)
   Nicolas Schulz (hdr, psd)            Martin "SpartanJ" Golini (stbi_info)
   Jonathan Dummer (tga)                James "moose2000" Brown (iPhone PNG)
   Jean-Marc Lienher (gif)              Ben "Disch" Wenger (io callbacks)
   Tom Seddon (pic)                   Omar Cornut (1/2/4-bit PNG)
   Thatcher Ulrich (psd)               Nicolas Guillemot (vertical flip)
   Ken Miller (pgm, ppm)                Richard Mitton (16-bit PSD)
   github:urraka (animated gif)          Junggon Kim (PNM comments)
                              Daniel Gibson (16-bit TGA)
                              socks-the-fox (16-bit PNG)
                              Jeremy Sawicki (handle all ImageNet JPGs)
 Optimizations & bugfixes
   Fabian "ryg" Giesen
   Arseny Kapoulkine
   John-Mark Allen

 Bug & warning fixes
   Marc LeBlanc           David Woo         Guillaume George   Martins Mozeiko
   Christpher Lloyd        Jerry Jansson      Joseph Thomson     Phil Jordan
   Dave Moore              Roy Eltham         Hayaki Saito       Nathan Reed
   Won Chun                Luke Graham        Johan Duparc       Nick Verigakis
   the Horde3D community   Thomas Ruf         Ronny Chevalier    Baldur Karlsson
   Janez Zemva             John Bartholomew   Michal Cichon      github:rlyeh
   Jonathan Blow           Ken Hamada         Tero Hanninen      github:romigrou
   Laurent Gomila          Cort Stratton      Sergio Gonzalez    github:svdijk
   Aruelien Pocheville     Thibault Reuille   Cass Everitt       github:snagar
   Ryamond Barbiero        Paul Du Bois       Engin Manap        github:Zelex
   Michaelangel007@github  Philipp Wiesemann  Dale Weiler        github:grim210
   Oriol Ferrer Mesia      Josh Tobin         Matthew Gregan     github:sammyhw
   Blazej Dariusz Roszkowski                  Gregory Mullen     github:phprus
   Christian Floisand      Kevin Schmidt                         github:poppolopoppo
*/

#include ...
#include <stdio.h>
#include "sipl.h"


struct img* imgread(const char*filename){
   struct img * img1 = (struct img*) malloc(sizeof(struct img));
   int i, x, y, comp;
   unsigned char *pixels;

         pixels = stbi_load(filename, &x, &y, &comp, 3);
         if (!pixels) {
                  fprintf(stderr, "image read error: %s\n", stbi_failure_reason());
                  return img1;
         }


         //printf("comp:%d ", comp);
   img1->row = y;
   img1->col = x;
   img1-> r = (uint8_t *)malloc(sizeof(uint8_t) * y * x);
   img1-> g = (uint8_t *)malloc(sizeof(uint8_t) * y * x);
   img1-> b = (uint8_t *)malloc(sizeof(uint8_t) * y * x);
   unsigned char *r, *g, *b;
         r = img1 -> r;
         g = img1 -> g;
         b = img1 -> b;
```

```
  for( i = 0; i < y * x; i++){

      img1-> r[0] = pixels[0];
      img1-> g[0] = pixels[1];
      img1-> b[0] = pixels[2];
      /*
      printf("x:%d y:%d , r:%d %d ", i%x, i/x, img1->r[0], pixels[0]);
                  printf("g:%d %d", img1->g[0], pixels[1]);
                  printf("b:%d %d\n", img1->b[0], pixels[2]);*/
      pixels += 3;
      (img1->r) ++;
      (img1->g) ++;
      (img1->b) ++;
  }

  img1->r = r;
  img1->g = g;
  img1->b = b;


  return img1;

}
```

8.1.3 std_image_write.c

```
CREDITS:

  PNG/BMP/TGA
    Sean Barrett
  HDR
    Baldur Karlsson
  TGA monochrome:
    Jean-Sebastien Guay
  misc enhancements:
    Tim Kelsey
  TGA RLE
    Alan Hickman
  initial file IO callback implementation
    Emmanuel Julien
  JPEG
```

```
    Jon Olick (original jo_jpeg.cpp code)
    Daniel Gibson
  bugfixes:
    github:Chribba
    Guillaume Chereau
    github:jry2
    github:romigrou
    Sergio Gonzalez
    Jonas Karlsson
    Filip Wasil
    Thatcher Ulrich
    github:poppolopoppo
    Patrick Boettcher

#include ...
#include "sipl.h"

int imgwrite(struct img *mat, int fmt, const char *filename){

   int flag, i;

   unsigned char *data;
   unsigned char * copy;
   data = (unsigned char *)malloc(sizeof(unsigned char) * mat->col * mat->row * 3);
   copy = data;
   //get the original form
   for(i = 0; i < mat->col * mat->row; i++){
      data[0] = mat->r[0];
      data[1] = mat->g[0];
      data[2] = mat->b[0];

      data += 3;
      (mat->r)++;
      (mat->g)++;
      (mat->b)++;

   }
   mat->r -= mat->col * mat->row;
         mat->g -= mat->col * mat->row;
         mat->b -= mat->col * mat->row;
   data = copy;
   switch (fmt) {
         case 0:
                  flag = stbi_write_png(filename, mat->col, mat->row, 3, data, mat->col * 3);
			if (!flag)
				fprintf(stderr, "imgwrite: Error writing to '%s'\n", filename);
			break;
		case 1:
			flag = stbi_write_bmp(filename, mat->col, mat->row, 3, data);
			if (!flag)
				fprintf(stderr, "imgwrite: Error writing to '%s'\n", filename);
			break;
		default:
			fprintf(stderr, "imgwrite: Unsupported output format (%d)\n", fmt);
			flag = -1;
		}

		return flag;
}
```

8.1.4 utils.c

```
#include "sipl.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

struct img * imgCreate(int m, int n) {
	struct img* img1 = (struct img*) malloc(sizeof(struct img));
	img1 -> row = m;
	img1 -> col = n;
	img1->r = (unsigned char *) calloc(sizeof(unsigned char), m * n);
	img1->g = (unsigned char *) calloc(sizeof(unsigned char), m * n);
	img1->b = (unsigned char *) calloc(sizeof(unsigned char), m * n);
	return img1;
}

struct matrix * matCreate(int m, int n) {
	struct matrix* mat = (struct matrix*) malloc(sizeof(struct matrix));
	mat->row = m;
	mat->col = n;
	mat->data = (double *)calloc(sizeof(double), m * n);
	return mat;
}

struct matrix * initMatrix(double * mat_data_ptr, int m, int n) {
	struct matrix* mat = (struct matrix*) malloc(sizeof(struct matrix));
	mat->row = m;
	mat->col = n;
	mat->data = mat_data_ptr;
	return mat;
}

double matAccess(struct matrix * mat, int i, int j) {
	if (mat->row <= i) i = mat->row - 1;
	if (mat->col <= j) j = mat->col - 1;
	return mat->data[i * mat->col + j];
}

int matAssign(struct matrix * mat, int i, int j, double val) {
	if (mat->row <= i) i = mat->row - 1;
	if (mat->col <= j) j = mat->col - 1;
	mat->data[i * mat->col + j] = val;
	return 0;
}

struct pixel * pixAccess(struct img * img1, int i, int j) {
	struct pixel* pix = (struct pixel*) malloc(sizeof(struct pixel));
	if (img1->row <= i) i = img1->row - 1;
	if (img1->col <= j) j = img1->col - 1;
	pix->r = img1->r[i * img1->col + j];
	pix->g = img1->g[i * img1->col + j];
	pix->b = img1->b[i * img1->col + j];
	return pix;
}

int pixAssign(struct img * img1, int i, int j, struct pixel* pix) {
	if (img1->row <= i) i = img1->row - 1;
	if (img1->col <= j) j = img1->col - 1;
	img1->r[i * img1->col + j] = (unsigned char) (pix->r);
	img1->g[i * img1->col + j] = (unsigned char) (pix->g);
	img1->b[i * img1->col + j] = (unsigned char) (pix->b);
	return 0;
}

int imgAccess(struct img * img1, int i, int j, const char* s) {
        if (img1->row <= i) i = img1->row - 1;
        if (img1->col <= j) j = img1->col - 1;
   if(strcmp(s, "r") == 0)
           return (int)img1->r[i * img1->col + j];
   if(strcmp(s, "g") == 0)
           return (int)img1->g[i * img1->col + j];
   if(strcmp(s, "b") == 0)
           return (int)img1->b[i * img1->col + j];
        return 0;
}

int imgAssign(struct img * img1, int i, int j, const char* s, int val) {
        if (img1->row <= i) i = img1->row - 1;
        if (img1->col <= j) j = img1->col - 1;
        if (val > 255) {
                val = 255;
        } else if (val < 0) {
                val = 0;
        }
   if(strcmp(s, "r") == 0)
           img1->r[i * img1->col + j] = (unsigned char) (val);
   if(strcmp(s, "g") == 0)
           img1->g[i * img1->col + j] = (unsigned char) (val);
   if(strcmp(s, "b") == 0)
           img1->b[i * img1->col + j] = (unsigned char) (val);
        return 0;
}

int imgWidth(struct img* img1){

   return img1->col;
}

int matWidth(struct matrix *mat){
   return mat->col;
}

int imgHeight(struct img* img1){

   return img1->row;
}

int matHeight(struct matrix *mat){
   return mat->row;
```

```
}

double int2float(int n) {
	return (double) n;
}

int float2int (double n) {
	return (int) n;
}

double power (double x, double y) {
	return pow(x,y);
}
int printMatrix(struct matrix * mat) {
	int i, j;
	//printf("%d\n", mat->row);
	//printf("%d\n", mat->col);
	for (i = 0; i < mat->row; i++) {
		for (j = 0; j < mat->col; j++) {
			printf("%f",mat->data[i * mat->col + j]);
		}
		printf("\n");
	}
	return 0;
}

int printPixel (struct pixel * pix) {
	printf("%d,%d,%d\n", pix->r, pix->g, pix->b);
	return 0;
}

int changeGrey(struct img* image){
	int i;
	unsigned char avg;
	for(i = 0; i < image->col * image->row; i++){
		avg = (image->r[i] + image->g[i] + image->b[i]) / 3;
		image->r[i] = avg;
		image->g[i] = avg;
		image->b[i] = avg;

	}
	return 0;
/*	img1->r -= img1->col * img1->row;
	img1->g -= img1->col * img1->row;
```

```
        img1->b -= img1->col * img1->row;
        */
}


//rotate clockwise
int rotateImage(struct img* image){
        int y, x, i;
        const int width = image->col;
        const int height = image->row;
        unsigned char temp_matrix[height * width * 3];

        //use a new matrix to store
        for(i = 0; i < width * height; i++){
                temp_matrix[3 * i] = image->r[i];
                temp_matrix[3 * i + 1] = image->g[i];
                temp_matrix[3 * i + 2] = image->b[i];
        }



        for(y = 0; y < width; y++){
                for(x = 0; x < height; x++){
                        image->r[y * height + x] = temp_matrix[3 * ((height - x - 1) * width +
(width - y - 1))];
                        image->g[y * height + x] = temp_matrix[3 * ((height - x - 1) * width +
(width - y - 1)) + 1] ;
                        image->b[y * height + x] = temp_matrix[3 * ((height - x - 1) * width +
(width - y - 1)) + 2];
                }
        }

        image->row = width;
        image->col = height;
        return 0;
}



int flipImage(struct img* image, int n){
        if(n != -1 && n!= 1) return 1;
        int x, y;
        unsigned char tmpr, tmpg, tmpb;
        if(n == 1){
                for(y = 0; y < image -> row; y++){
                        for(x = 0; x < image -> col / 2; x++){
                                tmpr = image->r[y * image -> col + x];
                                image->r[y * image -> col + x] = image->r[y * image -> col +
image -> col - x - 1];
                                image->r[y * image -> col + image -> col - x - 1] = tmpr;

                                tmpg = image->g[y * image -> col + x];
                                image->g[y * image -> col + x] = image->g[y * image -> col +
image -> col - x - 1];
                                image->g[y * image -> col + image -> col - x - 1] = tmpg;

                                tmpb = image->b[y * image -> col + x];
                                image->b[y * image -> col + x] = image->b[y * image -> col +
image -> col - x - 1];
                                image->b[y * image -> col + image -> col - x - 1] = tmpb;
                        }
                }
        }

        else {
                for(x = 0; x < image -> col; x++){
                        for(y = 0; y < image -> row / 2; y++){
                                tmpr = image->r[y * image -> col + x];
                                image->r[y * image -> col + x] = image->r[(image -> row - y -
1) * image -> col + x];
                                image->r[(image -> row - y - 1) * image -> col + x] = tmpr;

                                tmpg = image->g[y * image -> col + x];
                                image->g[y * image -> col + x] = image->g[(image -> row - y -
1) * image -> col + x];
                                image->g[(image -> row - y - 1) * image -> col + x] = tmpg;

                                tmpb = image->b[y * image -> col + x];
                                image->b[y * image -> col + x] = image->b[(image -> row - y -
1) * image -> col + x];
                                image->b[(image -> row - y - 1) * image -> col + x] = tmpb;
                        }
                }
        }

        return 0;
}
```

```
//create kernel matrix
struct matrix * kernel(const char* str){
          int i;
          struct matrix* mat = (struct matrix*) malloc(sizeof(struct matrix));

          if(strcmp("edge", str) == 0){
                    mat->data = (double *)malloc(sizeof(double) * 9);
                    for(i = 0; i < 9; i++){
                              if( i == 4) mat->data[i] = 8;
                              else mat->data[i] = -1;
                    }

          mat->col = mat->row = 3;
          }

          else if(strcmp("sharpen", str) == 0){
                    mat->data = (double *)malloc(sizeof(double) * 9);
                    for(i = 0; i < 9; i++){
                              if(i == 4) mat->data[i] = 5;
                              else if(i % 2 == 1) mat->data[i] = -1;
                              else mat->data[i] = 0;
                    }

                    mat->col = mat->row = 3;
          }

          else if(strcmp("gaussian", str) == 0){
                    mat->data = (double *)malloc(sizeof(double) * 9);

                    for(i = 0; i < 9; i++){
                              if(i == 4) mat->data[i] = 0.25;
                              else if(i % 2 == 1) mat->data[i] = 0.125;
                              else mat->data[i] = 1/16;
                    }

                    mat->col = mat->row = 3;
          }

          return mat;

}
```

```
struct img * convImg(struct img *image, struct matrix *mat){

        int x, y;
        int i, j;
        const int width = image->col;
        const int height = image->row;

        //rows and cols of matrix must be odd
        if(mat->col % 2 == 0 || mat->row % 2 == 0) return image;


        double* temp_mat; //[height * width * 3] = {0};
        temp_mat = (double *)malloc(sizeof(double) * 3 * height * width);
        memset(temp_mat, 0, 3 * height * width * sizeof(double));



        //save the result in tmp matrix
        for(y = 0; y < image->row; y++)       {

                //if it is at the edge of picture
                if( y - mat->row / 2 < 0 || y + mat->row / 2 >= image->row)   continue;

                for(x = 0; x < image->col; x++){

                        if( x - mat->col / 2 < 0 || x + mat->col / 2 >= image->col) continue;

                        for( i = -mat->row / 2; i <= mat->row / 2; i++){
                                for(j = - mat->col / 2; j <= mat->col / 2; j++){
                                        temp_mat[ 3 * (y * width + x)]                  +=
(double)image->r[(y + i) * width + x + j] * mat->data[(i + mat->row / 2) * mat->col + j +
mat->col / 2];
                                        temp_mat[ 3 * (y * width + x) + 1]    +=
(double)image->g[(y + i) * width + x + j] * mat->data[(i + mat->row / 2) * mat->col + j +
mat->col / 2];
                                        temp_mat[ 3 * (y * width + x) + 2]    +=
(double)image->b[(y + i) * width + x + j] * mat->data[(i + mat->row / 2) * mat->col + j +
mat->col / 2];

                                }
                        }

                }
```

```
	}

	//give the value to image
	for(y = 0; y < image->row; y++)	{

		//if it is at the edge of picture
		if( y - mat->row / 2 < 0 || y + mat->row / 2 >= image->row)   continue;
		for(x = 0; x < image->col; x++){

			if( x - mat->col / 2 < 0 || x + mat->col / 2 >= image->col) continue;
			for(i = 0; i < 3; i++){
				if(temp_mat[3 * (y * width + x) + i] > 255 ) temp_mat[3 * (y *
width + x) + i] = 255;
				else if (temp_mat[3 * (y * width + x) + i] < 0 ) temp_mat[3 * (y
* width + x) + i] = 0;
			}

			image->r[y * width + x] = (unsigned char) temp_mat[3 * (y * width +
x)];
			image->g[y * width + x] = (unsigned char) temp_mat[3 * (y * width +
x) + 1];
			image->b[y * width + x] = (unsigned char) temp_mat[3 * (y * width +
x) + 2];
		}
	}

	free(temp_mat);

	return image;
}

struct img* imgAdd(struct img *img1,struct img *img2 ){
	struct img* img3;
	img3 = (struct img *)malloc(sizeof(struct img));
	img3->r = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
	img3->g = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
	img3->b = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
	img3->col = img1->col;
	img3->row = img1->row;

	int i = 0;
	//unsigned char tmp;
	for(i = 0; i < img1->col * img1->row; i++){
		if((int)img1->r[i] + (int)img2 -> r[i] <= 255)
```

```
                        img3->r[i] = img1->r[i] + img2 -> r[i];
                else
                        img3->r[i] = 255;
                if((int)img1->g[i] + (int)img2 -> g[i] <= 255)
                        img3->g[i] = img1->g[i] + img2 -> g[i];
                else
                        img3->g[i] = 255;
                if((int)img1->b[i] + (int)img2 -> b[i] <= 255)
                        img3->b[i] = img1->b[i] + img2 -> b[i];
                else
                        img3->b[i] = 255;



        }


        return img3;
}


struct img* imgCopy(struct img *img1){
        struct img* img3;
        img3 = (struct img *)malloc(sizeof(struct img));
        img3->r = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img3->g = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img3->b = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img3->col = img1->col;
        img3->row = img1->row;

        int i = 0;
        //unsigned char tmp;
        for(i = 0; i < img1->col * img1->row; i++){

                img3->r[i] = img1->r[i];

                img3->g[i] = img1->g[i];

                img3->b[i] = img1->b[i];

        }

        return img3;
}
```

```
struct img* elementAdd(struct img *img1, int num2 ){
        /*struct img* img1;
        img3 = ( img *)malloc(sizeof(img3));
        img1->r = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img1->g = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img1->b = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img1->col = img1->col;
        img1->row = img1->row;
        */
        unsigned char num;
        num = (unsigned char) num2;
        // printf("num: %d \n");
        int i = 0;
        //unsigned char tmp;
        for(i = 0; i < img1->col * img1->row; i++){

                //printf("int r %d  sum:%d", (int)(img1->r[i]), (int)(img1->r[i]) + num2 );
                //printf("int g %d  sum:%d", (int)(img1->g[i]), (int)(img1->g[i]) + num2 );
                //printf("int b %d  sum:%d", (int)(img1->b[i]), (int)(img1->b[i]) + num2 );

                if((img1->r[i]) + num2 <= 255)
                        img1->r[i] = img1->r[i] + num;
                else{
                        //printf("r %d \n", i);
                        img1->r[i] = 255;
                }


                if((img1->g[i]) + num2 <= 255)
                        img1->g[i] = img1->g[i] + num;
                else{
                        //printf("g %d \n", i);
                        img1->g[i] = 255;
                }


                if((img1->b[i]) + num2 <= 255)
                        img1->b[i] = img1->b[i] + num;
                else{
                        //printf("b %d \n", i);
                        img1->b[i] = 255;
                }
        }

        //printf("address %d  r : %d", img1, img1->r);
        return img1;
}


struct img* elementSub(struct img *img1, int num2 ){
        /*struct img* img3;
        img3 = (struct img *)malloc(sizeof(img3));
        img3->r = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img3->g = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img3->b = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
        img3->col = img1->col;
        img3->row = img1->row;
        */
        int i = 0;
        unsigned char num = (unsigned char)num2;
        //unsigned char tmp; struct
        for(i = 0; i < img1->col * img1->row; i++){
                if(img1->r[i] - num2 > 0)
                        img1->r[i] = img1->r[i] - num;
                else
                        img1->r[i] = 0;

                if(img1->g[i] - num2 > 0)
                        img1->g[i] = img1->g[i] - num;
                else
                        img1->g[i] = 0;

                if(img1->b[i] - num2 > 0)
                        img1->b[i] = img1->b[i] - num;
                else
                        img1->b[i] = 0;

        }

        return img1;
}


struct img* elementMult(struct img *img1, double num ){
        /*
```

```
	struct img* img3;
	img3 = (struct img *)malloc(sizeof(img3));
	img3->r = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
	img3->g = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
	img3->b = (unsigned char *) malloc(sizeof(unsigned char) * img1->col * img1->row);
	img3->col = img1->col;
	img3->row = img1->row;
	*/
	int i = 0;
	//unsigned char tmp;
	for(i = 0; i < img1->col * img1->row; i++){

		img1->r[i] = (unsigned char) img1->r[i] * num;
		img1->g[i] = (unsigned char)img1->g[i] * num;
		img1->b[i] = (unsigned char)img1->b[i] * num;

	}
	return img1;
}

struct pixel * pixAddPix(struct pixel * pix1, struct pixel * pix2) {
	if (pix1->r + pix2->r < 255) {
		pix1->r = pix1->r + pix2->r;
	} else {
		pix1->r = 255;
	}
	if (pix1->g + pix2->g < 255) {
		pix1->g = pix1->g + pix2->g;
	} else {
		pix1->g = 255;
	}
	if (pix1->b + pix2->b < 255) {
		pix1->b = pix1->b + pix2->b;
	} else {
		pix1->b = 255;
	}
	return pix1;
}

struct pixel * pixAddInt(struct pixel * pix1, int val) {
	if (pix1->r + val > 255) {
		pix1->r = 255;
	} else if (pix1->r + val < 0) {
		pix1->r = 0;
	} else {
		pix1->r = (unsigned char) pix1->r + val;
	}
	if (pix1->g + val > 255) {
		pix1->g = 255;
	} else if (pix1->g + val < 0) {
		pix1->g = 0;
	} else {
		pix1->g = (unsigned char) pix1->g + val;
	}
	if (pix1->b + val > 255) {
		pix1->b = 255;
	} else if (pix1->b + val < 0) {
		pix1->b = 0;
	} else {
		pix1->b = (unsigned char) pix1->b + val;
	}
	return pix1;
}

struct pixel * pixSubInt(struct pixel * pix1, int val) {
	if (pix1->r - val > 255) {
		pix1->r = 255;
	} else if (pix1->r - val < 0) {
		pix1->r = 0;
	} else {
		pix1->r = (unsigned char) pix1->r - val;
	}
	if (pix1->g - val > 255) {
		pix1->g = 255;
	} else if (pix1->g - val < 0) {
		pix1->g = 0;
	} else {
		pix1->g = (unsigned char) pix1->g - val;
	}
	if (pix1->b - val > 255) {
		pix1->b = 255;
	} else if (pix1->b - val < 0) {
		pix1->b = 0;
	} else {
		pix1->b = (unsigned char) pix1->b - val;
	}
	return pix1;
}
```

```
struct pixel * pixMultFloat(struct pixel * pix1, double val) {
	if (pix1->r * val > 255) {
		pix1->r = 255;
	} else if (pix1->r * val < 0) {
		pix1->r = 0;
	} else {
		pix1->r = (unsigned char) pix1->r * val;
	}
	if (pix1->g * val > 255) {
		pix1->g = 255;
	} else if (pix1->g * val < 0) {
		pix1->g = 0;
	} else {
		pix1->g = (unsigned char) pix1->g * val;
	}
	if (pix1->b * val > 255) {
		pix1->b = 255;
	} else if (pix1->b * val < 0) {
		pix1->b = 0;
	} else {
		pix1->b = (unsigned char) pix1->b * val;
	}
	return pix1;
}
```

8.2 Our own library

8.2.1 stdlib.sp

```
Matrix sliceMat(Matrix mat, int left, int up, int cols, int rows) {
   int i;
   int j;
   int down;
   int right;
   Matrix res;
   down = up + rows;
   right = left + cols;

   if(down >= mat.height){
        down = mat.height;
    }
    if(right >= mat.width){
        right = mat.width;
    }

    res = matCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left] = mat[i][j];
        }
    }
    return res;
}

int main(){

    Matrix res;
    Matrix mat;

    res = [1.0,2.0,3.0,4.0,5.0;
           6.0,7.0,8.0,9.0,10.0;
           11.0,2.0,3.0,7.0,8.0;
            1.0,2.0,3.0,4.0,7.0;
            2.0,3.0,4.0,8.0,8.0];


    mat = sliceMat(res, 1, 1, 2, 2);

    printMatrix(mat);
}


Matrix sliceMat(Matrix mat, int left, int up, int cols, int rows) {
    int i;
    int j;
    int down;
    int right;
    Matrix res;
    down = up + rows;
    right = left + cols;

    if(down >= mat.height){
        down = mat.height;
    }
    if(right >= mat.width){
        right = mat.width;
    }

    res = matCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left] = mat[i][j];
        }
    }
    return res;
}

int main(){

    Matrix res;
    Matrix mat;

    res = [1.0,2.0,3.0,4.0,5.0;
           6.0,7.0,8.0,9.0,10.0;
           11.0,2.0,3.0,7.0,8.0;
            1.0,2.0,3.0,4.0,7.0;
            2.0,3.0,4.0,8.0,8.0];


    mat = sliceMat(res, 1, 1, 2, 2);

    printMatrix(mat);
}
```

8.2.2 sipllib.sp

```
Image resize (Image img, float scaleX, float scaleY){
    float cols;
    float rows;
    float bp_row;
    float bp_col;
    int i;
    int j;
    float delx;
    float dely;
    Pixel tmp;
    int tmp2;
    int indexy;
    int indexx;
    Image res;

    cols =  img.width * scaleX;
    rows =  img.height * scaleY;

    res = imgCreate(float2int(rows), float2int(cols));

    for(i = 0; i < float2int(rows); i = i + 1){
       for(j = 0; j < float2int(cols); j = j + 1){
          bp_row =  i / scaleY;
          bp_col =  j / scaleX;
          indexx = float2int(bp_col);
          indexy = float2int(bp_row);

          delx = bp_col -  indexx;
          dely = bp_row -  indexy;

          tmp =  img[indexy][indexx]["a"] * (1.0 - delx) * (1.0 - dely);
          tmp = tmp +  img[indexy + 1][indexx]["a"] * delx * ( 1.0 - dely) ;
          tmp = tmp +  img[indexy + 1][indexx + 1]["a"] * dely * delx;
          tmp = tmp +  img[indexy][indexx + 1]["a"] * (1.0 - delx) * dely;
          res[i][j]["a"] = tmp;

       }
    }

    return res;
}

Matrix gaussian(int size, float sigma){

    int i;
    int j;
    float e;
    float tmp;
    Matrix gauss;
    float sum;

    gauss = matCreate(size, size);
    e = 2.718281828459;
    for(i = 0; i < size; i = i + 1){
        for(j = 0; j < size; j = j + 1){
            tmp = 0 - (power((i - (size - 1)/2.0), 2.0) + power((j - (size - 1)/2.0), 2.0))/(2.0 * sigma
* sigma);
            gauss[i][j] = power(e, tmp);
            sum = sum + gauss[i][j];
        }
    }

    for(i = 0; i < size; i = i + 1){
        for(j = 0; j < size; j = j + 1){
            gauss[i][j] = gauss[i][j]/sum;
        }
    }
    return gauss;
}

Image sliceImg(Image img, int left, int up, int cols, int rows) {
    int i;
    int j;
    int down;
    int right;
    Image res;
    down = up + rows;
    right = left + cols;

    if(down >= img.height){
        down = img.height;
    }
    if(right >= img.width){
        right = img.width;
    }

    res = imgCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left]["r"] = img[i][j]["r"];
            res[i - up][j - left]["g"] = img[i][j]["g"];
            res[i - up][j - left]["b"] = img[i][j]["b"];
        }
    }
    return res;
}

Matrix sliceMat(Matrix mat, int left, int up, int cols, int rows) {
    int i;
    int j;
    int down;
    int right;
    Matrix res;
    down = up + rows;
    right = left + cols;

    if(down >= mat.height){
        down = mat.height;
    }
    if(right >= mat.width){
        right = mat.width;
    }

    res = matCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left] = mat[i][j];
        }
    }
    return res;
}
```

8.3 ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

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

type uop = Neg | Not
type typ = Int | Bool | Float | Void | Image | String | Matrix | Pixel


type bind = typ * string

type expr =
    Literal of int
  | BoolLit of bool
  | FloatLit of float
  | MatrixLit of expr list list
  | Id of string
  | StringLit of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | MatAccess of string * expr * expr
  | MatAssign of string * expr * expr * expr
  | ImgAccess of string * expr * expr * expr
  | ImgAssign of string * expr * expr * expr * expr
  | GetWidth of string
  | GetHeight of string
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list
```

```
(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Conv -> "**"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | FloatLit(f) -> string_of_float f
  | StringLit(s)   -> s
  | MatrixLit(_) -> "matrix [" ^ "]" (* TODO *)
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | MatAccess(v, row, col) -> v ^ "[ " ^ string_of_expr row ^ "][ " ^ string_of_expr col ^ " ]"
  | MatAssign(v, row, col, e) -> v ^ "[ " ^ string_of_expr row ^ "][ " ^ string_of_expr col ^ " ] =
" ^ string_of_expr e
  | ImgAssign(v, row, col, chan, e) -> v ^ "[ " ^ string_of_expr row ^ "][ " ^ string_of_expr col
^ " ][ " ^ string_of_expr chan ^ "] = " ^ string_of_expr e
  | ImgAccess(v, row, col, chan) -> v ^ "[ " ^ string_of_expr row ^ "][ " ^ string_of_expr col ^ "
][ " ^ string_of_expr chan ^ "]"
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | GetWidth (s)   -> s
  | GetHeight (s)   -> s
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | Image -> "image"
  | Matrix -> "matrix"
  | String -> "string"
  | Pixel -> "pixel"


let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.4 codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let llctx = L.global_context () in
  let lib_buffer = L.MemoryBuffer.of_file "lib/utils.bc" in
  let llm = Llvm_bitreader.parse_bitcode llctx lib_buffer in
  let the_module = L.create_module context "Sipl"
  and i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context
  and i1_t   = L.i1_type   context
  and void_t = L.void_type context
  and string_t = L.pointer_type (L.i8_type context)
  and float_t = L.double_type context
  and image_t = L.pointer_type (match L.type_by_name llm "struct.img" with
    None -> raise (Failure "struct.img doesn't defined.")
  | Some x -> x)
  and matrix_t = L.pointer_type (match L.type_by_name llm "struct.matrix" with
    None -> raise (Failure "struct.matrix doesn't defined.")
  | Some x -> x)
  and pixel_t = L.pointer_type (match L.type_by_name llm "struct.pixel" with
    None -> raise (Failure "struct.pixel doesn't defined.")
  | Some x -> x)
in
```

```
let ltype_of_typ = function
    A.Int -> i32_t
  | A.Bool -> i1_t
  | A.Image -> image_t
  | A.Matrix -> matrix_t
  | A.Pixel -> pixel_t
  | A.Void -> void_t
  | A.Float -> float_t
  | A.String -> string_t in

(* Declare each global variable; remember its value in a map *)
let global_vars =
  let global_var m (t, n) =
    let init = L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module, t) m in
  List.fold_left global_var StringMap.empty globals in

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in

(* Declare the built-in printbig() function *)
let printbig_t = L.function_type i32_t [| i32_t |] in
let printbig_func = L.declare_function "printbig" printbig_t the_module in

(* Declare built-in functions*)
let printImage_t = L.function_type i32_t [| image_t |] in
let printImage_func = L.declare_function "printImage" printImage_t the_module in

let printMatrix_t = L.function_type i32_t [| matrix_t |] in
let printMatrix_f = L.declare_function "printMatrix" printMatrix_t the_module in

let printPixel_t = L.function_type i32_t [| pixel_t |] in
let printPixel_f = L.declare_function "printPixel" printPixel_t the_module in

 (* Maths ----------------------------------------------------------*)

let int2float_t = L.function_type float_t [| i32_t |] in
let int2float_f = L.declare_function "int2float" int2float_t the_module in

let float2int_t = L.function_type i32_t [| float_t |] in
let float2int_f = L.declare_function "float2int" float2int_t the_module in

let power_t = L.function_type float_t [| float_t; float_t |] in
let power_f = L.declare_function "power" power_t the_module in

let matAccess_t = L.function_type float_t [| matrix_t; i32_t; i32_t |] in
let matAccess_f = L.declare_function "matAccess" matAccess_t the_module in

let matAssign_t = L.function_type float_t [| matrix_t; i32_t; i32_t; float_t |] in
let matAssign_f = L.declare_function "matAssign" matAssign_t the_module in

let imgAccess_t = L.function_type i32_t [| image_t; i32_t; i32_t; string_t;|] in
let imgAccess_f = L.declare_function "imgAccess" imgAccess_t the_module in

let imgAssign_t = L.function_type i32_t [| image_t; i32_t; i32_t; string_t; i32_t|] in
let imgAssign_f = L.declare_function "imgAssign" imgAssign_t the_module in

let pixAccess_t = L.function_type pixel_t [| image_t; i32_t; i32_t|] in
let pixAccess_f = L.declare_function "pixAccess" pixAccess_t the_module in

let pixAssign_t = L.function_type i32_t [| image_t; i32_t; i32_t; pixel_t|] in
let pixAssign_f = L.declare_function "pixAssign" pixAssign_t the_module in

let initMatrix_t      = L.function_type matrix_t [| L.pointer_type i8_t; i32_t; i32_t |] in
let initMatrix_func   = L.declare_function "initMatrix" initMatrix_t the_module in

let imgWidth_t = L.function_type i32_t [| image_t |] in
let imgWidth_f = L.declare_function "imgWidth" imgWidth_t the_module in

let matWidth_t = L.function_type i32_t [| matrix_t |] in
let matWidth_f = L.declare_function "matWidth" matWidth_t the_module in

let imgHeight_t = L.function_type i32_t [| image_t |] in
let imgHeight_f = L.declare_function "imgHeight" imgHeight_t the_module in

let matHeight_t = L.function_type i32_t [| matrix_t |] in
let matHeight_f = L.declare_function "matHeight" matHeight_t the_module in

let kernel_t = L.function_type matrix_t [| string_t |] in
let kernel_f = L.declare_function "kernel" kernel_t the_module in

let imgread_t = L.function_type image_t [| string_t |] in
let imgread_func = L.declare_function "imgread" imgread_t the_module in

let imgwrite_t = L.function_type i32_t [| image_t; i32_t; string_t |] in
let imgwrite_func = L.declare_function "imgwrite" imgwrite_t the_module in
```

```
  let convImg_t = L.function_type image_t [| image_t; matrix_t |] in
  let convImg_f = L.declare_function "convImg" convImg_t the_module in

  let changeGrey_t = L.function_type i32_t [| image_t |] in
  let changeGrey_f = L.declare_function "changeGrey" changeGrey_t the_module in

  let rotateImage_t = L.function_type i32_t [| image_t |] in
  let rotateImage_f = L.declare_function "rotateImage" rotateImage_t the_module in

  let flipImage_t = L.function_type i32_t [| image_t; i32_t |] in
  let flipImage_f = L.declare_function "flipImage" flipImage_t the_module in



  let elementAdd_t = L.function_type image_t [| image_t; i32_t |] in
  let elementAdd_f = L.declare_function "elementAdd" elementAdd_t the_module in

  let elementSub_t = L.function_type image_t [| image_t; i32_t |] in
  let elementSub_f = L.declare_function "elementSub" elementSub_t the_module in

  let elementMult_t = L.function_type image_t [| image_t; float_t |] in
  let elementMult_f = L.declare_function "elementMult" elementMult_t the_module in

  let imgAdd_t = L.function_type image_t [| image_t; image_t |] in
  let imgAdd_f = L.declare_function "imgAdd" imgAdd_t the_module in

  let imgCopy_t = L.function_type image_t [| image_t |] in
  let imgCopy_f = L.declare_function "imgCopy" imgCopy_t the_module in

  let imgCreate_t = L.function_type image_t [| i32_t; i32_t |] in
  let imgCreate_f = L.declare_function "imgCreate" imgCreate_t the_module in

  let matCreate_t = L.function_type matrix_t [| i32_t; i32_t |] in
  let matCreate_f = L.declare_function "matCreate" matCreate_t the_module in

(*          Pixel Functions *)

  let pixAddPix_t = L.function_type pixel_t [| pixel_t; pixel_t |] in
  let pixAddPix_f = L.declare_function "pixAddPix" pixAddPix_t the_module in

  let pixAddInt_t = L.function_type pixel_t [| pixel_t; i32_t |] in
  let pixAddInt_f = L.declare_function "pixAddInt" pixAddInt_t the_module in

  let pixSubInt_t = L.function_type pixel_t [| pixel_t; i32_t |] in
  let pixSubInt_f = L.declare_function "pixSubInt" pixSubInt_t the_module in

(*   let pixMultInt_t = L.function_type pixel_t [| pixel_t; i32_t |] in
  let pixMultInt_f = L.declare_function "pixMultInt" pixMultInt_t the_module in *)

  let pixMultFloat_t = L.function_type pixel_t [| pixel_t; float_t |] in
  let pixMultFloat_f = L.declare_function "pixMultFloat" pixMultFloat_t the_module in

  (* Define each function (arguments and return type) so we can call it *)
  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types =
            Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
      in let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
    let string_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
    let float_format_str = L.build_global_stringptr "%.3f\n" "fmt" builder in

    (* Construct the function's "locals": formal arguments and locally
       declared variables.  Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the "locals" map *)
    let local_vars =
      let add_formal m (t, n) p = L.set_value_name n p;
            let local = L.build_alloca (ltype_of_typ t) n builder in
            ignore (L.build_store p local builder);
            StringMap.add n (local, t) m in

      let add_local m (t, n) =
            let local_var = L.build_alloca (ltype_of_typ t) n builder
            in StringMap.add n (local_var, t) m in

      let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.A.locals in
```

```
(* Return the value for a variable or formal argument *)
let lookup n = try StringMap.find n local_vars
               with Not_found -> StringMap.find n global_vars
in

let idx n m          = [| L.const_int i32_t n; L.const_int i32_t m |] in

let get_mptr m b =
    let arr_ptr = L.build_in_bounds_gep m (idx 0 0) "build_in_bounds_gep" b in
     L.build_bitcast arr_ptr (L.pointer_type i8_t) "mat_ptr" b
in

let build_matrix l expr builder =
    let rows = List.length l in
    let cols = List.length (List.hd l) in
    let flat_list = List.fold_left List.append [] l in
    let list_of_const = List.map fst (List.map (expr builder) flat_list) in
    let flat_array = Array.of_list list_of_const in
    let mat = L.const_array float_t flat_array in
    let var = L.build_alloca (L.array_type float_t (cols * rows)) "mat" builder in
    ignore (L.build_store mat var builder);
    L.build_call initMatrix_func [| (get_mptr var builder); (L.const_int i32_t rows);
                          (L.const_int i32_t cols) |] "initMatrix" builder
in

let get_string = function
  A.StringLit s -> s
  | _ -> raise (Failure ("Get String of a none string"))

in

(* Construct code for an expression; return its value *)
let rec expr builder = function
        A.Literal i -> (L.const_int i32_t i, A.Int)
  | A.StringLit s -> (L.build_global_stringptr s s builder, A.String)
  | A.BoolLit b -> (L.const_int i1_t (if b then 1 else 0), A.Bool)
  | A.FloatLit f -> (L.const_float float_t f, A.Float)
  | A.MatrixLit l -> (build_matrix l expr builder, A.Matrix)
  | A.Noexpr -> (L.const_int i32_t 0, A.Void)
  | A.Id s -> (L.build_load (fst (lookup s)) s builder, snd (lookup s))
  | A.Binop (e1, op, e2) ->
          let (e1', typ1) = expr builder e1
          and (e2', typ2) = expr builder e2 in
          (match op with
          A.Add     ->
          (match typ1 with
             A.Int    ->
                (match typ2 with
                   A.Int     -> (L.build_add e1' e2' "tmp" builder, typ1)
                 | A.Float   -> (L.build_fadd (L.build_call int2float_f [| e1' |] "int2float"
builder) e2' "tmp" builder, A.Float)
                 | A.Image   -> (L.build_call elementAdd_f [| e2' ; e1' |] "elementAdd" builder,
typ2)
                 | A.Pixel   -> (L.build_call pixAddInt_f [| e2' ; e1' |] "pixAddInt" builder, typ2)
                 | _         -> raise (Failure (" unsupported operation "))
                )
           | A.Float   ->
                (match typ2 with
                   A.Float     -> (L.build_fadd e1' e2' "tmp" builder, typ1)
                 | A.Int   -> (L.build_fadd e1' (L.build_call int2float_f [| e2' |] "int2float"
builder) "tmp" builder, A.Float)
                 | _         -> raise (Failure (" unsupported operation "))
                )
           | A.Image   ->
                (match typ2 with
                   A.Int     -> (L.build_call elementAdd_f [| e1' ; e2' |] "elementAdd" builder,
typ1)
                 | A.Image   -> (L.build_call imgAdd_f [| e1' ; e2' |] "elementAdd" builder,
typ1)
                 | _         -> raise (Failure (" unsupported operation "))
                )
           | A.Pixel   ->
                (match typ2 with
                   A.Int     -> (L.build_call pixAddInt_f [| e1' ; e2' |] "pixAddInt" builder, typ1)
                 | A.Pixel   -> (L.build_call pixAddPix_f [| e1' ; e2' |] "pixAddPix" builder,
typ1)
                 | _         -> raise (Failure (" unsupported operation "))
                )
           | _         -> raise (Failure (" unsupported operation "))
          )
        | A.Sub     ->
          (match typ1 with
             A.Int    ->
                (match typ2 with
                   A.Int     -> (L.build_sub e1' e2' "tmp" builder, typ1)
                 | A.Float   -> (L.build_fsub (L.build_call int2float_f [| e1' |] "int2float" builder)
e2' "tmp" builder, A.Float)
                 | _         -> raise (Failure (" unsupported operation "))
                )

            | A.Float   ->
                (match typ2 with
                   A.Float     -> (L.build_fsub e1' e2' "tmp" builder, typ1)
                  | A.Int   -> (L.build_fsub e1' (L.build_call int2float_f [| e2' |] "int2float"
builder) "tmp" builder, A.Float)
                  | _         -> raise (Failure (" unsupported operation "))
                )

            | A.Image   ->
                (match typ2 with
                   A.Int     -> (L.build_call elementSub_f [| e1' ; e2' |] "elementSub" builder,
typ1)
                  | _         -> raise (Failure (" unsupported operation "))
                )
            | A.Pixel   ->
                (match typ2 with
                   A.Int     -> (L.build_call pixSubInt_f [| e1' ; e2' |] "pixSubInt" builder, typ1)
                  | _         -> raise (Failure (" unsupported operation "))
                )
            | _         -> raise (Failure (" unsupported operation "))
           )
         | A.Mult    ->
           (match typ1 with
              A.Int     ->
                (match typ2 with
                   A.Int     -> (L.build_mul e1' e2' "tmp" builder, A.Int)
                  | A.Float   -> (L.build_fmul (L.build_call int2float_f [| e1' |] "int2float"
builder) e2' "tmp" builder, A.Float)
                  | _         -> raise (Failure (" unsupported operation "))
                )
            | A.Float   ->
              (match typ2 with
                   A.Float   -> (L.build_fmul e1' e2' "tmp" builder, typ1)
                  | A.Int     -> (L.build_fmul e1' (L.build_call int2float_f [| e2' |] "int2float"
builder) "tmp" builder, A.Float)
                  | A.Image   -> (L.build_call elementMult_f [| e2' ; e1' |] "elementMult" builder,
typ2)
                  | A.Pixel   -> (L.build_call pixMultFloat_f [| e2' ; e1' |] "pixMultFloat" builder,
typ2)
                  | _         -> raise (Failure (" unsupported operation "))
              )
            | A.Image   ->
```

```
                  (match typ2 with
                       A.Float     -> (L.build_call elementMult_f [| e1' ; e2' |] "elementMult"
builder, typ1)
                     | _         -> raise (Failure (" unsupported operation "))
                  )
              | A.Pixel   ->
                  (match typ2 with
                       A.Float     -> (L.build_call pixMultFloat_f [| e1' ; e2' |] "pixMultFloat"
builder, typ1)
                     | _         -> raise (Failure (" unsupported operation "))
                  )
              | _         -> raise (Failure (" unsupported operation "))
            )
          | A.Div     ->
            (match typ1 with
               A.Int     ->
                  (match typ2 with
                       A.Int     -> (L.build_sdiv e1' e2' "tmp" builder, A.Int)
                     | A.Float   -> (L.build_fdiv (L.build_call int2float_f [| e1' |] "int2float" builder)
e2' "tmp" builder, A.Float)
                     | _         -> raise (Failure (" unsupported operation "))
                   )

              | A.Float   ->
                  (match typ2 with
                       A.Float     -> (L.build_fdiv e1' e2' "tmp" builder, A.Float)
                     | A.Int   -> (L.build_fdiv e1' (L.build_call int2float_f [| e2' |] "int2float"
builder) "tmp" builder, A.Float)
                     | _         -> raise (Failure (" unsupported operation "))
                   )
              | _         -> raise (Failure (" unsupported operation "))
            )
           | A.And     -> (L.build_and e1' e2' "tmp" builder, A.Bool)
           | A.Or      -> (L.build_or e1' e2' "tmp" builder, A.Bool)
           | A.Equal   ->
            (match typ1 with
               A.Int     -> (L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder, A.Bool)
              | A.Float   -> (L.build_fcmp L.Fcmp.Ueq e1' e2' "tmp" builder, A.Bool)
              | _         -> raise (Failure (" unsupported operation "))
            )

           | A.Neq     ->
            (match typ1 with
               A.Int     -> (L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder, A.Bool)
          | A.Float   -> (L.build_fcmp L.Fcmp.Une e1' e2' "tmp" builder, A.Bool)
          | _         -> raise (Failure (" unsupported operation "))
        )

      | A.Less    ->
        (match typ1 with
           A.Int     ->
             (match typ2 with
                  A.Int -> (L.build_icmp L.Icmp.Slt e1' e2' "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Ult (L.build_call int2float_f [| e1' |]
"int2float" builder) e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | A.Float   ->
             (match typ2 with
                  A.Int -> (L.build_fcmp L.Fcmp.Ult e1' (L.build_call int2float_f [| e2' |]
"int2float" builder) "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Ult e1' e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )


          | _         -> raise (Failure (" unsupported operation "))
        )
      | A.Leq     ->
        (match typ1 with
           A.Int     ->
             (match typ2 with
                  A.Int -> (L.build_icmp L.Icmp.Sle e1' e2' "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Ule (L.build_call int2float_f [| e1' |]
"int2float" builder) e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | A.Float   ->
             (match typ2 with
                  A.Int -> (L.build_fcmp L.Fcmp.Ule e1' (L.build_call int2float_f [| e2' |]
"int2float" builder) "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Ule e1' e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | _         -> raise (Failure (" unsupported operation "))
        )
      | A.Greater ->
        (match typ1 with
           A.Int     ->
             (match typ2 with
                  A.Int -> (L.build_icmp L.Icmp.Sgt e1' e2' "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Ugt (L.build_call int2float_f [| e1' |]
"int2float" builder) e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | A.Float   ->
             (match typ2 with
                  A.Int -> (L.build_fcmp L.Fcmp.Ugt e1' (L.build_call int2float_f [| e2' |]
"int2float" builder) "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Ugt e1' e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | _        -> raise (Failure (" unsupported operation "))
        )
      | A.Geq     ->
        (match typ1 with
           A.Int     ->
             (match typ2 with
                  A.Int -> (L.build_icmp L.Icmp.Sge e1' e2' "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Uge (L.build_call int2float_f [| e1' |]
"int2float" builder) e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | A.Float   ->
           (match typ2 with
                  A.Int -> (L.build_fcmp L.Fcmp.Uge e1' (L.build_call int2float_f [| e2' |]
"int2float" builder) "tmp" builder, A.Bool)
                  | A.Float -> (L.build_fcmp L.Fcmp.Uge e1' e2' "tmp" builder, A.Bool)
                  | _ -> raise (Failure (" unsupported operation "))
             )
          | _        -> raise (Failure (" unsupported operation "))
        )
      | A.Conv  ->
        (match typ1 with
           A.Image -> ( match typ2 with
               A.Matrix   -> (L.build_call convImg_f [|e1' ; e2' |] "convImg" builder, A.Image)
               | _ ->raise (Failure("unsupport operation"))
           )

           | A.Matrix -> ( match typ2 with
               A.Image   -> (L.build_call convImg_f [|e2' ; e1' |] "convImg" builder, A.Image)
               | _ -> raise (Failure("unsupport operation"))
           )
           | _ -> raise (Failure("unsupport operation"))
        )
      )
| A.Unop(op, e) ->
      let (e', typ') = expr builder e in
      ((match op with
           A.Neg     -> L.build_neg
          | A.Not     -> L.build_not) e' "tmp" builder, typ')
| A.MatAccess (s, r, c) ->
      let row = fst (expr builder r) in
      let col = fst (expr builder c) in
      let mat = L.build_load (fst (lookup s)) s builder in
      (L.build_call matAccess_f [| mat; row; col |] "matAccess" builder , A.Float)
| A.MatAssign (s, r, c, e) ->
      let (e', typ') = expr builder e in
      let row = fst (expr builder r) in
      let col = fst (expr builder c) in
      let mat = L.build_load (fst (lookup s)) s builder in
      (match typ' with
        A.Float -> (L.build_call matAssign_f [| mat; row; col; e' |] "matAssign" builder ,
A.Int)
        | A.Int ->
        (L.build_call matAssign_f [| mat; row; col; (L.build_call int2float_f [| e' |] "int2float"
builder) |] "matAssign" builder , A.Int)
        | _ -> raise (Failure("unsupport Matrix Assign"))
      )

| A.ImgAccess (s, r, c, sr) ->
      let row = fst (expr builder r) in
      let col = fst (expr builder c) in
      let str = fst (expr builder sr) in
      let img = L.build_load (fst (lookup s)) s builder in
      let flag = get_string sr in
      if (flag = "a") then (L.build_call pixAccess_f [| img; row; col |] "pixAccess" builder ,
A.Pixel)
      else (L.build_call imgAccess_f [| img; row; col; str |] "imgAccess" builder , A.Int)


| A.ImgAssign (s, r, c, sr, e) ->
      let e' = fst (expr builder e) in
      let row = fst (expr builder r) in
      let col = fst (expr builder c) in
      let str = fst (expr builder sr) in
```

```
            let img = L.build_load (fst (lookup s)) s builder in
            let flag = get_string sr in
            if (flag = "a") then (L.build_call pixAssign_f [| img; row; col; e' |] "pixAssign" builder
, A.Int)
            else (L.build_call imgAssign_f [| img; row; col; str; e' |] "imgAssign" builder , A.Int)
    | A.GetWidth (id) ->
          let (s, typ) =  lookup id in
          let id' = L.build_load s id builder in
            (match typ with
              A.Image -> (L.build_call imgWidth_f [| id'|] "img" builder, A.Int)
              | A.Matrix -> (L.build_call matWidth_f [| id'|] "mat" builder, A.Int)
              | _ -> raise (Failure("unsupport operation"))
              )

    | A.GetHeight (id) ->
          let (s, typ) =  lookup id in
          let id' = L.build_load s id builder in
            (match typ with
              A.Image -> (L.build_call imgHeight_f [| id'|] "img" builder, A.Int)
              | A.Matrix -> (L.build_call matHeight_f [| id'|] "mat" builder, A.Int)
              | _ -> raise (Failure("unsupport operation"))
              )
    | A.Assign (s, e) -> let (e', typ') = expr builder e and (l, ltyp) = lookup s in
            (match (ltyp, typ') with
              (A.Float, A.Int) ->
                  ignore (L.build_store (L.build_call int2float_f [| e' |] "int2float" builder) l builder);
(e', ltyp)
              | _ -> ignore (L.build_store e' l builder ); (e', typ')
            )

    | A.Call ("print", [e]) | A.Call ("printb", [e]) ->
                  (L.build_call printf_func [| int_format_str ; fst (expr builder e) |] "printf" builder
, A.Int)
    | A.Call ("prints", [e]) ->
            (L.build_call printf_func [| string_format_str ; fst (expr builder e) |] "printf" builder,
A.Int)
    | A.Call ("printfloat", [e]) ->
            (L.build_call printf_func [| float_format_str ; fst (expr builder e) |] "printf" builder,
A.Int)
    | A.Call ("printbig", [e]) ->
                  (L.build_call printbig_func [| fst (expr builder e) |] "printbig" builder, A.Int)
    | A.Call ("printImage", [e]) ->
            (L.build_call printImage_func [| fst (expr builder e) |] "printImage" builder, A.Int)
    | A.Call ("printMatrix", [e]) ->
```

```
          (L.build_call printMatrix_f [| fst (expr builder e) |] "printMatrix" builder, A.Int)
     | A.Call ("printPixel", [e]) ->
          (L.build_call printPixel_f [| fst (expr builder e) |] "printPixel" builder, A.Int)
     | A.Call ("int2float", [e]) ->
          (L.build_call int2float_f [| fst (expr builder e) |] "int2float" builder , A.Float)
     | A.Call ("float2int", [e]) ->
          (L.build_call float2int_f [| fst (expr builder e) |] "float2int" builder , A.Int)
     | A.Call ("power", [e1; e2]) ->
          (L.build_call power_f [| fst (expr builder e1); fst (expr builder e2) |] "power" builder ,
A.Float)
     | A.Call ("kernel", [e]) ->
          (L.build_call kernel_f [| fst (expr builder e) |] "kernel" builder, A.Matrix)
     | A.Call ("matCreate", [e1; e2]) ->
          (L.build_call matCreate_f [| fst (expr builder e1); fst (expr builder e2) |] "matCreate"
builder, A.Matrix)
     | A.Call ("changeGrey", [img]) ->
          (L.build_call changeGrey_f [| fst (expr builder img) |] "changeGrey" builder, A.Int)
     | A.Call ("rotateImage", [img]) ->
          (L.build_call rotateImage_f [| fst (expr builder img) |] "rotateImage" builder, A.Int)
     | A.Call ("flipImage", [img; flag]) ->
          (L.build_call flipImage_f [| fst (expr builder img); fst (expr builder flag)|] "flipImage"
builder, A.Int)
     | A.Call ("imgCopy", [img]) ->
          (L.build_call imgCopy_f [| fst (expr builder img) |] "imgCopy" builder, A.Image)
     | A.Call ("imgCreate", [e1; e2]) ->
          (L.build_call imgCreate_f [| fst (expr builder e1); fst (expr builder e2) |] "imgCreate"
builder, A.Image)
     | A.Call ("imgread", [filename]) ->
          (L.build_call imgread_func [| fst (expr builder filename)|] "imgread" builder,
A.Image)
     | A.Call ("imgwrite", [img; flag; filename]) ->
          (L.build_call imgwrite_func [| fst (expr builder img); fst (expr builder flag); fst (expr
builder filename) |] "imgwrite" builder, A.Int)
     | A.Call (f, act) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
          let actuals = List.rev (List.map fst (List.map (expr builder) (List.rev act))) in
          let result = (match fdecl.A.typ with A.Void -> ""
                                  | _ -> f ^ "_result") in
        (L.build_call fdef (Array.of_list actuals) result builder, fdecl.A.typ)
   in



   (* Invoke "f builder" if the current block doesn't already
   have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
       Some _ -> ()
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor *)
let rec stmt builder = function
       A.Block sl -> List.fold_left stmt builder sl
  | A.Expr e -> ignore (expr builder e); builder
  | A.Return e -> ignore (match fdecl.A.typ with
               A.Void -> L.build_ret_void builder
                 | _ -> L.build_ret (fst (expr builder e)) builder); builder
  | A.If (predicate, then_stmt, else_stmt) ->
     let bool_val = fst (expr builder predicate) in
        let merge_bb = L.append_block context "merge" the_function in

        let then_bb = L.append_block context "then" the_function in
        add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          (L.build_br merge_bb);

        let else_bb = L.append_block context "else" the_function in
        add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
          (L.build_br merge_bb);

        ignore (L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

  | A.While (predicate, body) ->
         let pred_bb = L.append_block context "while" the_function in
         ignore (L.build_br pred_bb builder);

         let body_bb = L.append_block context "while_body" the_function in
         add_terminal (stmt (L.builder_at_end context body_bb) body)
           (L.build_br pred_bb);

         let pred_builder = L.builder_at_end context pred_bb in
         let bool_val = fst (expr pred_builder predicate) in

         let merge_bb = L.append_block context "merge" the_function in
         ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
         L.builder_at_end context merge_bb
```

```
    | A.For (e1, e2, e3, body) -> stmt builder
            ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
  in

  (* Build the code for each statement in the function *)
  let builder = stmt builder (A.Block fdecl.A.body) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.A.typ with
      A.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
 in

 List.iter build_function_body functions;
 the_module
```

8.5 parser.mly

```
/* Ocamlyacc parser for Sipl */

%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA LBRACKET RBRACKET
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT MPLUS MMINUS MTIMES CONV
DOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID IMAGE STRING MATRIX
FLOAT WIDTH HEIGHT PIXEL


%token <int> LITERAL
%token <float> FLOAT_LITERAL
%token <string> ID
%token <string> STRING_LIT
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right NOASSIGN
%right ASSIGN
```

```
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left DOT
%left MMINUS MPLUS
%left MTIMES
%left CONV
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }


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

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { typ = $1;
           fname = $2;
           formals = $4;
           locals = List.rev $7;
           body = List.rev $8 } }

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

formal_list:
    typ ID                  { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | BOOL { Bool }
  | FLOAT { Float }
  | VOID { Void }
  | IMAGE { Image }
  | MATRIX { Matrix }
  | STRING  { String }
  | PIXEL {Pixel}

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

vdecl:
   typ ID SEMI { ($1, $2) }

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

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    LITERAL          { Literal($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | FLOAT_LITERAL    { FloatLit($1) }
  | ID               { Id($1) }
  | STRING_LIT       { StringLit($1)}
  | expr CONV   expr { Binop($1, Conv,  $3) }
  | 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 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 AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET %prec NOASSIGN {
MatAccess($1, $3, $6)}
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET ASSIGN expr    {
MatAssign($1, $3, $6, $9)}
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET LBRACKET expr
RBRACKET %prec NOASSIGN { ImgAccess($1, $3, $6, $9)}
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET LBRACKET expr
RBRACKET ASSIGN expr { ImgAssign($1, $3, $6, $9, $12)}
  | ID DOT WIDTH { GetWidth($1)}
  | ID DOT HEIGHT { GetHeight($1) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LBRACKET lit_mat RBRACKET   { MatrixLit($2) }
  | LPAREN expr RPAREN { $2 }

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

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

lit_mat:
  lit_mat_rev              { List.rev $1 }

lit_mat_rev:
  lit_list                 { [$1] }
  | lit_mat_rev SEMI lit_list   { $3 :: $1 }

lit_list:
  lit_list_rev             { List.rev $1}
```

```
lit_list_rev:
   lit                  { [$1] }
  | lit_list_rev COMMA lit  { $3 :: $1 }

lit:
   LITERAL              { Literal($1) }
  | FLOAT_LITERAL          { FloatLit($1) }
```

8.6 scanner.mll

```
(* Ocamllex scanner for Sipl *)

{ open Parser }

let digit = ['0'-'9']

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }          (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['         { LBRACKET }
| ']'         { RBRACKET }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '.'      { DOT }
| ".*"              { MTIMES    }
| ".+"              { MPLUS     }
| ".-"             { MMINUS    }
| "**"              { CONV      }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "width"  { WIDTH }
| "height" { HEIGHT}
| "=="     { EQ }
| "!="     { NEQ }
| '<'     { LT }
| "<="     { LEQ }
| ">"     { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"     { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "float"  { FLOAT }
| "bool"   { BOOL }
| "Image"  { IMAGE }
| "Matrix" { MATRIX }
| "Pixel" { PIXEL }
| "string" { STRING }
| "void"   { VOID }
| "true"   { TRUE }
| "false"  { FALSE }

| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| '-'?(digit+) ['.'] digit+ as lxm { FLOAT_LITERAL(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| '"'([^'"']* as lxm)'"'  {STRING_LIT(lxm)}
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

8.7 semant.ml

```
(* Semantic checking for the Sipl compiler *)
```

```
open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
        n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet == rvaluet || lvaluet == Float && rvaluet == Int then lvaluet else raise err
  in

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

  (**** Checking Functions ****)

  if List.mem "print" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print may not be defined")) else ();

  report_duplicate (fun n -> "duplicate function " ^ n)
```

```
   (List.map (fun fd -> fd.fname) functions);

  (* Function declaration for a named function *)

  let built_in_func_list = [
  { typ = Image; fname = "imgread"; formals = [(String, "x")]; locals = []; body = []};
  { typ = Void; fname = "imgwrite"; formals = [(Image, "img"); (Int, "x"); (String,
"filename")]; locals = []; body = []};
  { typ = Image; fname = "imgCopy"; formals = [(Image, "img")]; locals = []; body = []};
  { typ = Image; fname = "imgCreate"; formals = [(Int, "x"); (Int, "x")]; locals = []; body = []};
  { typ = Matrix; fname = "matCreate"; formals = [(Int, "x"); (Int, "x")]; locals = []; body = []};
  { typ = Matrix; fname = "kernel"; formals = [(String, "x")]; locals = []; body = []};
  { typ = Float; fname = "int2float"; formals = [(Int, "x")]; locals = []; body = []};
  { typ = Float; fname = "power"; formals = [(Float, "x"); (Float, "y")]; locals = []; body = []};
  { typ = Int; fname = "float2int"; formals = [(Float, "x")]; locals = []; body = []};
  { typ = Image; fname = "convImg"; formals = [ (Image, "img"); (Matrix, "mat")]; locals = [];
body = []};
  { typ = Void; fname = "changeGrey"; formals = [ (Image, "img") ]; locals = []; body = []};
  { typ = Void; fname = "rotateImage"; formals = [ (Image, "img") ]; locals = []; body = []};
  { typ = Void; fname = "flipImage"; formals = [ (Image, "img"); (Int, "x") ]; locals = []; body
= []};
  { typ = Void; fname = "printMatrix"; formals = [(Matrix, "mat")]; locals = []; body = []};
  { typ = Void; fname = "printPixel"; formals = [(Pixel, "mat")]; locals = []; body = []};
  { typ = Void; fname = "prints"; formals = [(String, "x")]; locals = []; body = []};
  { typ = Void; fname = "printfloat"; formals = [(Float, "x")]; locals = []; body = []};
  { typ = Void; fname = "printImage"; formals = [(Image, "x")]; locals = []; body = []};
  { typ = Void; fname = "print"; formals = [(Int, "x")]; locals = []; body = []};
  { typ = Void; fname = "printb"; formals = [(Bool, "x")]; locals = []; body = []};
  { typ = Void; fname = "printbig"; formals = [(Int, "x")]; locals = []; body = []}
  ] in

  let built_in_decls =  List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
    StringMap.empty built_in_func_list

   in

  let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                         built_in_decls functions
  in

  let function_decl s = try StringMap.find s function_decls
      with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in
```

```
let _ = function_decl "main" in (* Ensure "main" is defined *)

let check_function func =

  List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
    " in " ^ func.fname)) func.formals;

  report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
    (List.map snd func.formals);

  List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
    " in " ^ func.fname)) func.locals;

  report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
    (List.map snd func.locals);

  (* Type of each variable (global, formal, or local *)
  let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (globals @ func.formals @ func.locals )
  in

  let type_of_identifier s =
    try StringMap.find s symbols
    with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in

  (* Return the type of an expression or throw an exception *)
  let rec expr = function
        Literal _ -> Int
    | BoolLit _ -> Bool
    | StringLit _ -> String
    | FloatLit _ -> Float
    | MatrixLit l ->
      let col = List.length (List.hd l) in
      let matched = List.for_all (fun li -> (List.length li) == col) l in
    (match matched with
    | true -> Matrix
    | false -> raise (Failure ("Matrix Format Not Match"))
    )
    | MatAccess(s, r, c) -> let t0 = type_of_identifier s and t1 = expr r and t2 = expr c in
        (match t0 with
          Matrix when t1 = Int && t2 = Int -> Float
          | Matrix -> raise (Failure ("Matrix index should be Integer"))
          | _ -> raise (Failure ("identifier is not matrix"))
        )
    | MatAssign(s, r, c, e) ->
          let t0 = type_of_identifier s and t1 = expr r
              and t2 = expr c and t3 = expr e in
        (match t0 with
         Matrix when t1 = Int && t2 = Int && (t3 = Float || t3 = Int) -> Int
         | Matrix when t1 = Int && t2 = Int -> raise (Failure ("Matrix element value should be
Float"))
         | Matrix -> raise (Failure ("Matrix index should be Integer"))
         | _ -> raise (Failure ("identifier is not matrix"))
        )
    | ImgAssign(s, r, c, ch, e) ->
          let t0 = type_of_identifier s and t1 = expr r
              and t2 = expr c and t3 = expr e in
        (match t0 with
         Image when t1 = Int && t2 = Int && t3 = Int ->
           (match ch with
             | Id(_) -> Int
             | StringLit l when l = "r" || l = "g" || l = "b" -> Int
             | _ -> raise (Failure ("specified channel is not valid"))
           )
         | Image when t1 = Int && t2 = Int && t3 = Pixel ->
         (match ch with
             | Id(_) -> Pixel
             | StringLit l when l = "a" -> Pixel
             | _ -> raise (Failure ("specified channel is not valid"))
           )
         | Image when t1 = Int && t2 = Int -> raise (Failure ("Image element value should be
Integer"))
         | Image -> raise (Failure ("Image index should be Integer"))
         | _ -> raise (Failure ("identifier is not Image"))
        )
    | ImgAccess(s, r, c, ch) ->
          let t0 = type_of_identifier s and t1 = expr r
              and t2 = expr c in
        (match t0 with
         Image when t1 = Int && t2 = Int ->
         (match ch with
             | Id(_) -> Int
             | StringLit l when l = "r" || l = "g" || l = "b" -> Int
             | StringLit l when l = "a" -> Pixel
             | _ -> raise (Failure ("specified channel is not valid"))
         )
         | Image -> raise (Failure ("Image index should be Integer"))
```

```
          | _ -> raise (Failure ("identifier is not Image"))
        )
    | GetWidth(id) -> let typ = type_of_identifier id in
          (match typ with
          Matrix -> Int
          | Image -> Int
          | _ -> raise(Failure("identifier is not matrix nor image"))
          )
    | GetHeight(id) -> let typ = type_of_identifier id in
          (match typ with
          Matrix -> Int
          | Image -> Int
          | _ -> raise(Failure("identifier is not matrix nor image"))
          )

    | Id s -> type_of_identifier s
    | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
          (match op with
        Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
      | Add | Sub | Mult | Div when t1 = Float && (t2 = Int || t2 = Float) -> Float
      | Add | Sub | Mult | Div when t1 = Int && t2 = Float -> Float
      | Add | Div when t1 = Image && t2 = Int || t1 = Int && t2 = Image || t1 = Image && t2 =
Image -> Image
      | Add when t1 = Pixel && t2 = Int || t1 = Int && t2 = Pixel || t1 = Pixel && t2 = Pixel ->
Pixel
      | Sub when t1 = Image && t2 = Int -> Image
      | Sub when t1 = Pixel && t2 = Int -> Pixel
      | Conv when t1 = Image && t2 = Matrix || t1 = Matrix && t2 = Image -> Image
      | Mult when t1 = Image && t2 = Float || t1 = Float && t2 = Image -> Image
      | Mult when t1 = Pixel && t2 = Float || t1 = Float && t2 = Pixel -> Pixel
          | Equal | Neq when t1 = t2 -> Bool
          | Less | Leq | Greater | Geq when (t1 = Int || t1 = Float) && (t2 = Float || t2 = Int) ->
Bool
          | And | Or when t1 = Bool && t2 = Bool -> Bool
      | _ -> raise (Failure ("illegal binary operator " ^
            string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
            string_of_typ t2 ^ " in " ^ string_of_expr e))
      )
    | Unop(op, e) as ex -> let t = expr e in
           (match op with
             Neg when t = Int -> Int
  | Neg when t = Float -> Float
           | Not when t = Bool -> Bool
        | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                             string_of_typ t ^ " in " ^ string_of_expr ex)))
  | Noexpr -> Void
  | Assign(var, e) as ex -> let lt = type_of_identifier var
                         and rt = expr e in
    check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                                   " = " ^ string_of_typ rt ^ " in " ^
                                   string_of_expr ex))
  | Call(fname, actuals) as call -> let fd = function_decl fname in
     if List.length actuals != List.length fd.formals then
       raise (Failure ("expecting " ^ string_of_int
         (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
     else
       List.iter2 (fun (ft, _) e -> let et = expr e in
          ignore (check_assign ft et
            (Failure ("illegal actual argument found " ^ string_of_typ et ^
            " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
         fd.formals actuals;
       fd.typ
in

let check_bool_expr e = if expr e != Bool
 then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
 else () in

(* Verify a statement or throw an exception *)
let rec stmt = function
        Block sl -> let rec check_block = function
       [Return _ as s] -> stmt s
     | Return _ :: _ -> raise (Failure "nothing may follow a return")
     | Block sl :: ss -> check_block (sl @ ss)
     | s :: ss -> stmt s ; check_block ss
     | [] -> ()
    in check_block sl
  | Expr e -> ignore (expr e)
  | Return e -> let t = expr e in if t = func.typ then () else
     raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                 string_of_typ func.typ ^ " in " ^ string_of_expr e))

  | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
  | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                       ignore (expr e3); stmt st
  | While(p, s) -> check_bool_expr p; stmt s
in
```

```
    stmt (Block func.body)

  in
  List.iter check_function functions
```

8.8 sipl.ml

```
(* Top-level of the Sipl compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

module StringMap = Map.Make(String)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./sipl.native [-a|-l|-c] [file.sp]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;
  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

8.9 testall.sh

```
#!/bin/sh

# Regression testing script for sipl
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the sipl compiler.  Usually "./sipl.native"
# Try "_build/sipl.native" if ocamlbuild was unable to create a symbolic link.
SIPL="./sipl.native"
#SIPL="_build/sipl.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
   echo "Usage: testall.sh [options] [.sp files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
         echo "FAILED"
         error=1
   fi
   echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
         SignalError "$1 differs"
         echo "FAILED $1 differs from $2" 1>&2
   }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
         SignalError "$1 failed on $*"
         return 1
   }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
   echo $* 1>&2
   eval $* && {
         SignalError "failed: $* did not report an error"
         return 1
   }
   return 0
}

Check() {
   error=0
   basename=`echo $1 | sed 's/.*\///
                         s/.sp//'`
   reffile=`echo $1 | sed 's/.sp$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."
   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""
  cat "lib/sipllib.sp" $1 > "catfile.sp"


   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
   Run "$SIPL" "catfile.sp" ">" "${basename}.ll" &&
   # Run "$SIPL" "$1" ">" "${basename}.ll" &&
   Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
   Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "lib/stb_image.o"
"lib/stb_image_write.o" "lib/utils.o" "-lm"&&
   Run "./${basename}.exe" > "${basename}.out" &&
   Compare ${basename}.out ${reffile}.out ${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
         if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
         fi
         echo "OK"
         echo "###### SUCCESS" 1>&2
   else
         echo "###### FAILED" 1>&2
         globalerror=$error
   fi
}

CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                          s/.sp//'`
   reffile=`echo $1 | sed 's/.sp$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""
```

```
    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$SIPL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
            if [ $keep -eq 0 ] ; then
                rm -f $generatedfiles
            fi
            echo "OK"
            echo "###### SUCCESS" 1>&2
    else
            echo "###### FAILED" 1>&2
            globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
            k) # Keep intermediate files
                keep=1
                ;;
            h) # Help
                Usage
                ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
```

```
   exit 1
fi

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

for file in $files
do
   case $file in
        *test-*)
           Check $file 2>> $globallog
           ;;
        *fail-*)
           CheckFail $file 2>> $globallog
           ;;
        *)
           echo "unknown file type $file"
           globalerror=1
           ;;
   esac
done

exit $globalerror
```

8.10 test

8.10.1 fail-img-access1.err

```
Fatal error: exception Failure("specified channel is not valid")
```

8.10.2 fail-img-access1.sp

```
int main () {
        Image img;
```

```
   img = imgread("lena.png");
          print(img[0][0]["x"]);
}
```

8.10.3 fail-img-access2.err

```
Fatal error: exception Failure("Image index should be Integer")
```

8.10.4 fail-img-access2.sp

```
int main () {
          Image img;
   img = imgread("lena.png");
          print(img[0][0.5]["x"]);
}
```

8.10.5 fail-img-access3.err

```
Fatal error: exception Failure("identifier is not Image")
```

8.10.6 fail-img-access3.sp

```
int main () {
          Image img;
          int i;
   img = imgread("lena.png");
          print(i[0][0]["x"]);
}
```

8.10.7 fail-img-assign1.err

```
Fatal error: exception Failure("specified channel is not valid")
```

8.10.8 fail-img-assign1.sp

```
int main () {
          Image img;
```

```
	img = imgread("lena.png");
	img[1][2]["x"] = 0;
}
```

8.10.9 fail-img-assign2.err

```
Fatal error: exception Failure("Image element value should be Integer")
```

8.10.10 fail-img-assign2.sp

```
int main () {
	Image img;
	img = imgread("lena.png");
	img[1][2]["r"] = 0.5;
}
```

8.10.11 fail-img-assign3.err

```
Fatal error: exception Failure("Image index should be Integer")
```

8.10.12 fail-img-assign3.sp

```
int main () {
	Image img;
	img = imgread("lena.png");
	img[1][2.0]["r"] = 1;
}
```

8.10.13 fail-img-assign4.err

```
Fatal error: exception Failure("identifier is not Image")
```

8.10.14 fail-img-assign4.sp

```
int main () {
	Image img;
	Matrix mat;
	img = imgread("lena.png");
	mat[1][2]["r"] = 1;
}
```

8.10.15 fail-matrix-access1.err

```
Fatal error: exception Failure("Matrix index should be Integer")
```

8.10.16 fail-matrix-access1.sp

```
int main () {
	Matrix mat;
	mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
	printfloat(mat[1][2.0]);
}
```

8.10.17 fail-matrix-access2.err

```
Fatal error: exception Failure("identifier is not matrix")
```

8.10.18 fail-matrix-access2.sp

```
int main () {
	Matrix mat;
	int fakeMat;
	mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
	printfloat(fakeMat[1][2]);
}
```

8.10.19 fail-matrix-assign1.err

```
Fatal error: exception Failure("Matrix index should be Integer")
```

8.10.20 fail-matrix-assign1.sp

```
int main () {
	Matrix mat;
```

```
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        mat[1][2.0] = 12.0;
        printMatrix(mat);
}
```

8.10.21 fail-matrix-assign2.err

```
Fatal error: exception Failure("identifier is not matrix")
```

8.10.22 fail-matrix-assign2.sp

```
int main () {
        Matrix mat;
        int fakeMat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        fakeMat[1][2] = 12.0;
        printMatrix(mat);
}
```

8.10.23 fail-matrix-assign3.err

```
Fatal error: exception Failure("Matrix element value should be Float")
```

8.10.24 fail-matrix-assign3.sp

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        mat[1][2] = mat;
        printMatrix(mat);
}
```

8.10.25 fail-matrix1.err

```
Fatal error: exception Failure("Matrix index should be Integer")
```

8.10.26 fail-matrix1.sp

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0; 7.0, 8.0, 9.0];
}
```

8.10.27 test-clamp1.out

```
1.0000002.0000003.000000
4.0000005.0000006.000000
7.0000008.00000010.000000
```

8.10.28 test-clamp1.sp

```
int main()
{
  Image img;
  Matrix mat;
  img = imgread("lena.png");
  mat = [1.0,2.0,3.0;4.0,5.0,6.0;7.0,8.0,9.0];
  mat[3][3] = 10.0;
  img[img.height][img.width]["r"] = 0;
  imgwrite(img, 0, "lena-clamp1.png");
  printMatrix(mat);
}
```

8.10.26 fail-matrix1.sp

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0; 7.0, 8.0, 9.0];
}
```

8.10.27 test-generate-gaussian.out

```
0.0029690.0133060.0219380.0133060.002969
0.0133060.0596340.0983200.0596340.013306
0.0219380.0983200.1621030.0983200.021938
0.0133060.0596340.0983200.0596340.013306
0.0029690.0133060.0219380.0133060.002969
```

8.10.28 test-generate-gaussian.sp

```
 /*
Matrix gaussian(int size, float sigma){

    int i;
    int j;
    float e;
    float tmp;
    Matrix gauss;
    float sum;

    gauss = matCreate(size, size);
    e = 2.718281828459;
    for(i = 0; i < size; i = i + 1){
        for(j = 0; j < size; j = j + 1){
            tmp = 0 - (power((i - (size - 1)/2.0), 2.0) + power((j - (size - 1)/2.0), 2.0))/(2.0 * sigma
* sigma);
            gauss[i][j] = power(e, tmp);
            sum = sum + gauss[i][j];
        }
    }

    for(i = 0; i < size; i = i + 1){
        for(j = 0; j < size; j = j + 1){
            gauss[i][j] = gauss[i][j]/sum;
        }
    }
    return gauss;
}*/

int main(){

    Matrix mat;

    mat = gaussian(5, 1.0);

    printMatrix(mat);
}
```

8.10.29 test-generate-gaussian2.out

```

```

8.10.30 test-generate-gaussian2.sp

```
/*
Matrix gaussian(int size, float sigma){

    int i;
    int j;
    float e;
    float tmp;
    Matrix gauss;
    float sum;

    gauss = matCreate(size, size);
    e = 2.718281828459;
    for(i = 0; i < size; i = i + 1){
        for(j = 0; j < size; j = j + 1){
            tmp = 0 - (power((i - (size - 1)/2.0), 2.0) + power((j - (size - 1)/2.0), 2.0))/(2.0 * sigma
* sigma);
            gauss[i][j] = power(e, tmp);
            sum = sum + gauss[i][j];
        }
    }

    for(i = 0; i < size; i = i + 1){
        for(j = 0; j < size; j = j + 1){
            gauss[i][j] = gauss[i][j]/sum;
        }
    }
    return gauss;
}*/

int main(){
    Image img;
    Matrix mat;
    img = imgread("lena.png");
    mat = gaussian(5, 1.0);
    img = img ** mat ** mat ** mat;
    imgwrite(img, 0, "lena-gaussian3.png");
}
```

8.10.31 test-getheight1.out

```
461
```

8.10.32 test-getheight1.sp

```
int main () {
          Image img;
   int x;
   img = imgread("lena.png");
   x = img.height;
   print(x);
}
```

8.10.33 test-getheight2.out

```
2
```

8.10.34 test-getheight2.sp

```
int main () {
          Matrix mat;
   int x;
   mat = [1.0 ,2.0 ,3.0 ;4.0 ,5.0 ,6.0];
   x = mat.height;
   print(x);
}
```

8.10.35 test-getwidth1.out

```
2
```

8.10.36 test-getwidth1.sp

```
int main () {
          Image img;
   int x;
   img = imgread("lena.png");
   x = img.width;
   print(x);
}
```

8.10.37 test-getwidth2.out

```
3
```

8.10.38 test-getwidth2.sp

```
int main () {
		Matrix mat;
	int x;
	mat = [1.0 ,2.0 ,3.0 ;4.0 ,5.0 ,6.0];
	x = mat.width;
	print(x);
}

		Matrix mat;
		mat = [1.0, 2.0, 3.0; 4.0, 5.0; 7.0, 8.0, 9.0];
}
```

8.10.39 test-final1.out

```

```

8.10.40 test-final1.sp

```
int main(){

    float scalex;
    float scaley;
    Matrix mat;
    Matrix kern;
    Image img;
    Image img2;
    Image img3;
    Image img4;

    img = imgread("10.jpg");
    img2 = imgread("9.jpg");


    scalex = int2float(img.width) / img2.width;
    scaley = int2float(img.height) / img2.height;

    kern = gaussian(3, 0.5);

    img = img ** kern;


    img2 = resize(img2, 0.95, 0.95);
    img2 = sliceImg(img2, 0, 0 ,img.height, img.width );
    img2 = img * 0.45 + img2 * 0.55;

    imgwrite(img2, 0, "overlapg.png");

}
```

8.10.41 test-final2.out

```

```

8.10.42 test-final2.sp

```

int main(){

    Matrix kern;
    Image img;
    Image img2;

    img = imgread("3.jpg");


    kern = gaussian(3, 0.5);

    img2 = img ** kern;

    imgwrite(img, 0, "gauss1.png");

    kern = gaussian(3, 1.0);
    img2 = img ** kern;

    imgwrite(img, 0, "gauss2.png");

    kern = gaussian(5, 1.0);
    img2 = img ** kern;
    imgwrite(img, 0, "gauss3.png");

}
```

8.10.43 test-float-op1.out

```
<
>
<=
>=
```

8.10.44 test-float-op1.sp

```
int main()
{
  float i;
  i = 1.5;

  if (i < 2 && 1 < i) {
    prints("<");
  }

  if (i > 1 && 2 > i) {
    prints(">");
  }

  if (i <= 2 && 1 <= i) {
    prints("<=");
  }
  if (i >= 1 && 2 >= i) {
    prints(">=");
  }

  return 0;
}
```

8.10.45 test-float1.out

```
1.200
```

8.10.46 test-float1.sp

```
int main()
{
  float i;
  i = 1.2;
  printfloat(i);
  return 0;
}
```

8.10.47 test-float2.out

```
3.400
-1.000
0.400
3.600
```

8.10.48 test-float2.sp

```
int main()
{
  float i;
  i = 1.2 + 2.2;
  printfloat(i);
  i = 1.2 - 2.2;
  printfloat(i);
  i = 1.2 / 3.0;
  printfloat(i);
  i = 1.2 * 3.0;
  printfloat(i);
  return 0;
}
```

8.10.49 test-float3.out

```
>
<
>=
<=
==
```

8.10.50 test-float3.sp

```
int main()
{
  float i;
  i = 1.1;
  if (i > 1.0) {
    prints(">");
  }

  if (i < 1.2) {
    prints("<");
  }

  if (i >= 1.1) {
    prints(">=");
  }

  if (i <= 1.1) {
    prints("<=");
  }

  if (i == 1.1) {
    prints("==");
  }
  return 0;
}
```

8.10.51 test-float2int.out

```
1
```

8.10.52 test-float2int.sp

```
int main()
{
  int i;
  float j;
  j = 1.2;
  i = float2int(j);
  print(i);
  return 0;
}
```

8.10.53 test-floatfor.out

```
0.000
1.000
2.000
3.000
4.000
5.000
6.000
7.000
8.000
9.000
```

8.10.54 test-floatfor.sp

```
int main(){

   float i;

   for(i = 0.0; i < 10.0; i = i + 1.0){
      printfloat(i);
   }
}
```

8.10.55 test-floatwhile.out

```

```

8.10.56 test-floatwhile.out

```
int main(){
   float i;

   i = 1.0;
   while( i < 10.0){
      i = i + 1.0;
   }

}
```

8.10.57 test-img-imgAdd.out

```
Fatal error: exception Failure("Image index should be Integer")
```

8.10.58 test-img-imgAdd.sp

```
int main () {
        Image img;
        img = imgread("lena.png");
        img = img + img;
        imgwrite(img, 0, "lena-imgAdd.png");
}
```

8.10.59 test-img-imgAdd.out

```

```

8.10.60 test-img-imgAdd.out

```
int main () {
        Image img;
        img = imgCreate(100, 100);
        imgwrite(img, 0, "lena-imgCreate.png");
}
```

8.10.61 test-img-imgAdd.out

```

```

8.10.62 test-img-imgAdd.out

```
int main () {
        Image img;
        img = imgCreate(100, 100);
        imgwrite(img, 0, "lena-imgCreate.png");
}
```

8.10.63 test-img-pixaccess1.out

```
201,113,77
```

8.10.64 test-img-pixaccess1.sp

```
int main () {
	Image img;
	Pixel pix;
   img = imgread("lena.png");
   printPixel(img[0][0]["a"]);
}
```

8.10.65 test-img-pixassign1.out

```
201,113,77
197,108,76
197,108,76
```

8.10.66 test-img-pixassign1.sp

```
int main () {
	Image img;
	Pixel pix;
   img = imgread("lena.png");
   printPixel(img[0][0]["a"]);
   pix = img[10][10]["a"];
   printPixel(pix);
   img[0][0]["a"] = pix;
   printPixel(img[0][0]["a"]);
}
```

8.10.67 test-img-rotateImage.out

```
```

8.10.68 test-img-rotateImage.sp

```
int main () {
	Image img;
	img = imgread("lena.png");
	rotateImage(img);
	imgwrite(img, 0, "lena-rotate.png");
}
```

8.10.69 test-img-slice.out

```
```

8.10.70 test-img-slice.sp

```
/*
Image sliceImg(Image img, int left, int up, int cols, int rows) {
    int i;
    int j;
    int down;
    int right;
    Image res;
    down = up + rows;
    right = left + cols;

    if(down >= img.height){
        down = img.height;
    }
    if(right >= img.width){
        right = img.width;
    }

    res = imgCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left]["r"] = img[i][j]["r"];
            res[i - up][j - left]["g"] = img[i][j]["g"];
            res[i - up][j - left]["b"] = img[i][j]["b"];
        }
    }
    return res;
}
*/
int main(){

    Image res;
    Image img;

    res = imgread("lena.png");
    img = sliceImg(res, 150, 200, 100, 100);
    imgwrite(img, 0, "lena-slice.png");
}
```

8.10.71 test-img-slice2.out

```

```

8.10.72 test-img-slice2.sp

```
int main(){

    Image res;
    Image img;

    res = imgread("lena.png");
    img = sliceImg(res, 150, 200, 100, 100);

    imgwrite(img, 0, "lena-slice.png");
}
```

8.10.73 test-int2float.out

```
1.000
```

8.10.74 test-int2float.sp

```
int main()
{
  int i;
  float j;
  i = 1;
  j = int2float(i);
  printfloat(j);
  return 0;
}
```

8.10.75 test-int2float2.out

```
2.000
0.500
```

```
0.200
10.000
1.000
```

8.10.76 test-int2float2.sp

```
int main()
{
  int i;
  float j1;
  float j2;
  float j3;
  float j4;
  float j5;
  i = 1;
  j1 = i + 1.0;
  j2 = i - 0.5;
  j3 = i * 0.2;
  j4 = i / 0.1;
  j5 = 1;
  printfloat(j1);
  printfloat(j2);
  printfloat(j3);
  printfloat(j4);
  printfloat(j5);
  return 0;
}
```

8.10.77 test-int2float3.out

```
1.000
```

8.10.78 test-int2float3.sp

```
int main()
{
  Matrix mat;
  mat = matCreate(3,3);
  mat[1][1] = 1;
  printfloat(mat[1][1]);
  return 0;
}
```

8.10.79 test-kernel-edge.out

```
```

8.10.80 test-kernel-edge.sp

```
int main () {
        Matrix mat;
        Image img;
        mat = kernel("edge");
        img = imgread("lena.png");
        img = img**mat;
        imgwrite(img, 0, "lena-edge.png");
}
```

8.10.81 test-kernel-gaussian.out

```
```

8.10.82 test-kernel-gaussian.sp

```
int main () {
        Matrix mat;
        Image img;
        mat = kernel("gaussian");
        img = imgread("lena.png");
        img = mat ** img ** mat;
   imgwrite(img, 0, "lena-gaussian2.png");
}
```

8.10.83 test-kernel-sharpen.out

```
```

8.10.84 test-kernel-sharpen.sp

```
int main () {
        Matrix mat;
        Image img;
        mat = kernel("sharpen");
        img = imgread("lena.png");
        img ** mat;
    imgwrite(img, 0, "lena-sharpen.png");
}
```

8.10.85 test-local1.out

```
84
```

8.10.86 test-local1.sp

```
void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}

int main()
{
  foo(true);
  return 0;
}
```

8.10.87 test-local2.out

```
47
```

8.10.88 test-local2.sp

```
int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}
```

```
int main() {
 print(foo(37, false));
 return 0;
}
```

8.10.89 test-math1.out

```
4.000
```

8.10.90 test-math1.sp

```
int main()
{
  printfloat(power(2.0, 2.0));
}
```

8.10.91 test-matrix-access.out

```
6.000
```

8.10.92 test-matrix-access.sp

```
int main () {
	Matrix mat;
	int i;
	i = 2;
	mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
	printfloat(mat[1][i]);
}
```

8.10.93 test-matrix-assign.out

```
1.0000002.0000003.000000
4.0000005.00000012.000000
7.0000008.0000009.000000
```

8.10.94 test-matrix-assign.sp

```
int main () {
	Matrix mat;
	mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
	mat[1][2] = 12.0;
	printMatrix(mat);
}
```

8.10.95 test-matrix-assign0.sp

```
1.0000002.0000003.000000
4.0000005.0000006.000000
7.0000008.0000009.000000
```

8.10.96 test-matrix-assign0.sp

```
int main () {
	Matrix mat;
	mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
	printMatrix(mat);
}
```

8.10.97 test-image1.out

```

```

8.10.98 test-image1.sp

```
int main () {
	Image img;
   img = imgread("lena.png");
   img[0][3]["r"] = 0;


	imgwrite(img, 0, "lena-red-zero.png");
}
```

8.10.99 test-image2.out

```

```

8.10.100 test-image2.out

```
int main () {
          Image img;
   int i;
   img = imgread("lena.png");
   img[0][1]["r"] = 0;
   i = img[0][1]["r"];
   print(i);

}
```

8.10.101 test-image1.out

```

```

8.10.102 test-image1.sp

```
int main () {
          Image img;
   img = imgread("lena.png");
   img[0][3]["r"] = 0;


          imgwrite(img, 0, "lena-red-zero.png");
}
```

8.10.103 test-image2.out

```
0
```

8.10.104 test-image2.sp

```
int main () {
          Image img;
   int i;
   img = imgread("lena.png");
   img[0][1]["r"] = 0;
   i = img[0][1]["r"];
   print(i);
}
```

8.10.105 test-img-access1.out

```
201
```

8.10.106 test-img-access1.sp

```
int main () {
        Image img;
        int i;
   img = imgread("lena.png");
   i = 0;
        print(img[0][0]["r"]);
}
```

8.10.107 test-img-assign1.out

```

```

8.10.108 test-img-assign1.sp

```
int main () {
        Image img;
        int i;
   img = imgread("lena.png");
   i = 0;
   img[i][3 + 5]["r"] = 0;

        imgwrite(img, 0, "lena-red-zero.png");
}
```

8.10.109 test-img-changeGrey.out

```

```

8.10.110 test-img-changeGrey.sp

```
int main () {
        Image img;
        img = imgread("lena.png");
        changeGrey(img);
        imgwrite(img, 0, "lena-grey.png");
}
```

8.10.111 test-img-elementAdd.out

```

```

8.10.112 test-img-elementAdd.sp

```
int main () {
        Image img;
        img = imgread("lena.png");
        img = 20 + img + 20;
        imgwrite(img, 0, "lena-elementAdd.png");
}
```

8.10.113 test-img-elementMult.out

```

```

8.10.114 est-img-elementMult.sp

```
int main () {
        Image img;
        img = imgread("lena.png");
        img = img * 0.5;
        imgwrite(img, 0, "lena-elementMult.png");
}
```

8.10.115 test-img-elementMult2.out

```

```

8.10.116 test-img-elementMult2.sp

```
int main () {
        Image img;
        Image img2;
        Image img3;
        img = imgread("lena.png");
        img2 = imgCopy(img);
        img3 = img2 * 0.4 + 0.5 * img;
        imgwrite(img3, 0, "lena-elementMult2.png");
}
```

8.10.117 test-img-elementSub.out

```

```

8.10.118 test-img-elementSub.sp

```
int main () {
        Image img;
        img = imgread("lena.png");
        img = img - 20;
        imgwrite(img, 0, "lena-elementSub.png");
}
```

8.10.119 test-img-flip.out

```
1.000
```

8.10.120 test-img-flip.sp

```
int main () {
        Image img;
        img = imgread("lena.png");
        flipImage(img, 1);
        imgwrite(img, 0, "lena-flip1.png");
        flipImage(img, 1);
        imgwrite(img, 0, "lena-flip_1.png");
}
```

8.10.121 test-matrix-assign2.out

```
1.0000002.0000003.000000
4.0000005.0000009.000000
```

```
7.0000008.0000009.000000
```

8.10.122 test-matrix-assign2.sp

```
int main () {
        Matrix mat;
        mat = [1.0, 2.0, 3.0; 4.0, 5.0, 6.0; 7.0, 8.0, 9.0];
        mat[1][2] = mat[2][2];
        printMatrix(mat);
}
```

8.10.123 test-matrix-matCreate.out

```
0.0000000.0000000.000000
0.0000000.0000000.000000
0.0000000.0000000.000000
```

8.10.124 test-matrix-matCreate.sp

```
int main () {
        Matrix mat;
        mat = matCreate(3, 3);
        printMatrix(mat);
}
```

8.10.125 test-matrix-slice.out

```
7.0000008.000000
2.0000003.000000
```

8.10.126 test-matrix-slice.sp

```
/*
Matrix sliceMat(Matrix mat, int left, int up, int cols, int rows) {
   int i;
   int j;
   int down;
   int right;
   Matrix res;
   down = up + rows;
```

```
    right = left + cols;

    if(down >= mat.height){
        down = mat.height;
    }
    if(right >= mat.width){
        right = mat.width;
    }

    res = matCreate(down - up, right - left);

    for(i = up; i < down; i = i + 1){
        for(j = left; j < right; j = j + 1){
            res[i - up][j - left] = mat[i][j];
        }
    }
    return res;
}*/

int main(){

    Matrix res;
    Matrix mat;

    res = [1.0,2.0,3.0,4.0,5.0;
           6.0,7.0,8.0,9.0,10.0;
           11.0,2.0,3.0,7.0,8.0;
            1.0,2.0,3.0,4.0,7.0;
            2.0,3.0,4.0,8.0,8.0];


    mat = sliceMat(res, 1, 1, 2, 2);

    printMatrix(mat);
}
```

8.10.127 test-ops1.out

```
3
-1
2
50
99
```

```
0
1
99
1
0
99
1
0
99
1
1
0
99
0
1
99
0
1
1
```

8.10.128 test-ops1.sp

```
int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}
```

8.10.129 test-ops2.out

```
1
0
1
0
0
0
1
1
1
0
1
0
-10
42
```

8.10.130 test-ops2.sp

```
int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--42);
}
```

8.10.131 test-pix-ops1.out

```
201,113,77
203,115,79
50,28,19
40,18,9
```

8.10.132 test-pix-ops1.sp

```
int main()
{
  Pixel p1;
  Pixel p2;
  Image img;
  img = imgread("lena.png");
  p1 = img[0][0]["a"];
  printPixel(p1);
  p2 = 1 + p1 + 1;
  printPixel(p2);
  p2 = 0.5 * p1 * 0.5;
  printPixel(p2);
  p2 = p1 - 10;
  printPixel(p2);
  return 0;
}
```

8.10.133 test-overlap.out

```

```

8.10.134 test-overlap.sp

```
int main(){
    Image img1;
    Image img2;
    Matrix mat;
    mat = kernel("edge");
    img1 = imgread("lena.png");
    img2 = imgread("lena.png");
    img2 ** mat;

    img2 = img2 * 0.5 + img1 * 0.5;
    imgwrite(img2, 0, "lena-img-overlap.png");
}
```

8.10.135 test-string.out

```
test string
test string
```

8.10.136 test-string.sp

```
int main()
{
  string str;
  str = "test string";
  prints("test string");
  prints(str);
  return 0;
}
```

8.10.137 test-resize.out

```

```

8.10.138 test-resize.sp

```
/*
Image resize (Image img, float scaleX, float scaleY){
    float cols;
    float rows;
    float bp_row;
    float bp_col;
    int i;
    int j;
    float delx;
    float dely;
    Pixel tmp;
    int tmp2;
    int indexy;
    int indexx;
    Image res;

    cols =  img.width * scaleX;
    rows =  img.height * scaleY;

    res = imgCreate(float2int(rows), float2int(cols));

    for(i = 0; i < float2int(rows); i = i + 1){
        for(j = 0; j < float2int(cols); j = j + 1){
            bp_row =  i / scaleY;
            bp_col =  j / scaleX;
            indexx = float2int(bp_col);
            indexy = float2int(bp_row);

            delx = bp_col -  indexx;
            dely = bp_row -  indexy;

            tmp =  img[indexy][indexx]["a"] * (1.0 - delx) * (1.0 - dely);
            tmp = tmp +  img[indexy + 1][indexx]["a"] * delx * ( 1.0 - dely) ;
            tmp = tmp +  img[indexy + 1][indexx + 1]["a"] * dely * delx;
            tmp = tmp +  img[indexy][indexx + 1]["a"] * (1.0 - delx) * dely;
            res[i][j]["a"] = tmp;

        }
    }

    return res;
}
*/


int main(){
    Image img;
    Image img2;

    img = imgread("lena.png");

    img2 = resize(img, 1.5, 2.5);
    imgwrite(img2, 0, "lena-resize2.png");
}
```

8.10.139 test-resize2.out

```

```

8.10.140 test-resize2.sp

```
/*
Image resize (Image img, float scaleX, float scaleY){
    float cols;
    float rows;
    float bp_row;
    float bp_col;
    int i;
    int j;
    float delx;
    float dely;
    Pixel tmp;
    int tmp2;
    int indexy;
    int indexx;
    Image res;

    cols =  img.width * scaleX;
    rows =  img.height * scaleY;

    res = imgCreate(float2int(rows), float2int(cols));

    for(i = 0; i < float2int(rows); i = i + 1){
        for(j = 0; j < float2int(cols); j = j + 1){
            bp_row =  i / scaleY;
            bp_col =  j / scaleX;
            indexx = float2int(bp_col);
            indexy = float2int(bp_row);

            delx = bp_col -  indexx;
            dely = bp_row -  indexy;

            tmp =  img[indexy][indexx]["a"] * (1.0 - delx) * (1.0 - dely);
            tmp = tmp +  img[indexy + 1][indexx]["a"] * delx * ( 1.0 - dely) ;
            tmp = tmp +  img[indexy + 1][indexx + 1]["a"] * dely * delx;
            tmp = tmp +  img[indexy][indexx + 1]["a"] * (1.0 - delx) * dely;
            res[i][j]["a"] = tmp;

        }
    }

    return res;
}
*/


int main(){
    Image img;
    Image img2;

    img = imgread("lena.png");

    img2 = resize(img, 1.5, 2.5);

    imgwrite(img2, 0, "lena-resize2.png");
}
```

Reference:

TuSimple: An Easy Graph Language (JC)
http://www.cs.columbia.edu/~sedwards/classes/2017/4115-spring/index.html

DNAsharp: Molecular Biology Computation Language (DW)
http://www.cs.columbia.edu/~sedwards/classes/2016/4115-fall/index.html

CLAM: The Concise Linear Algebra Manipulation Language
http://www.cs.columbia.edu/~sedwards/classes/2017/4115-spring/

B. W. Kernighan and D. Ritchie. The C Programming Language

Dennis M. Ritchie. C Reference Manual