Colode

A Programming Language for Easy Image Manipulation

Members	**Uni**	**Position**
Steven Bonilla	sb3914	Manager
Tahmid Munat	tfm2109	Language Guru
Takuma Yasuda	ty2353	Language Guru
Willie Koomson	wvk2101	System Architect
Dimitri Borgers	djb2195	Tester

Contents

1 Introduction

1.1 Overview

Modifying and editing pictures professionally often requires a steep learning curve. While softwares such as Adobe Photoshop are quite known, it is often hard for a beginner to fully grasp it's power and expensive to purchase for infrequent usage. Languages such as C offer giant libraries and with proficient knowledge, can allow for extreme granularity. This tool, however, is mostly unreachable for non-programmers. A language that is simple to understand, easy to learn, and powerful enough to implement features such as image convolution is needed for proper adoption.

1.2 Purpose

Colode is an efficient, simple programming language used for manipulating individual or groups of pixel values. With a syntax that exploits the very best of both Python and Java, this language provides an extremely easy-to-learn and advanced way to modify image files for any use. The built-in functions can be used for the most common editing features available on other expert platforms. With the ability to control individual pixels on a loaded image file, a user will be able to alter anything at the most detailed level.

1.3 Use Cases

Image convolution allows for blurring, sharpening, embossing, edge detection, and more. With each image represented as a matrix, convolution is achieved through the process of adding each element of the image (the pixel values within the matrix) to its local neighbors. Through the built-in functions of Colode, a user can automatically calculate the height and width of a picture, and then choose to blur, change the color of, or increase the transparency of certain pixels by modifying the red, blue, green, and alpha channel values stored within the matrix.

2 Tutorial

2.1 Environment Setup

You need a version of Ocaml with the Map.Make.find_opt function, 4.0.5 or greater. Additionally, you must install ocamlbuild, ocamlfind, and the llvm bindings for ocaml.

2.1.1: Under OS X (High Sierra)

1. **Install Homebrew:**
   ```
   /usr/bin/ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
   ```
2. **Verify Homebrew is installed correctly:**
   ```
   brew doctor
   ```
3. **Install Opam:**
   ```
   $ brew install opam
   ```
4. **Configure Opam:**
   ```
   $ opam init
   ```
5. **Install LLVM**
   ```
   $ brew install llvm
   ```
   Take note of where brew places the llvm executables. It will show you the path to them under the CAVEATS section of the post-install terminal output.
6. **Setup Opam on Environment**
   ```
   $ eval `opam config env`
   ```
   Ensure that the version of llvm you install here matches the version you installed via brew.
7. **Install Libffi**
   ```
   $ brew install libffi
   ```
8. **Create a symbolic link to the lli command:**
   ```
   $ sudo ln -s /usr/local/opt/llvm/bin/lli /usr/bin/lli
   ```
9. **Install OCaml LLVM library**
   ```
   $ opam install llvm.5.0
   ```
10. **Install clang++**
    ```
    $ wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key | sudo apt-key
    add -
    $ sudo apt-get update
    $ sudo apt-add-repository "deb http://apt.llvm.org/xenial/
    llvm-toolchain-xenial-6.0 main"
    $ sudo apt-get install -y clang-6.0
    ```
11. **Install libpng**
    ```
    $ sudo apt-get install libpng12-0
    ```

2.1.2 Under Windows (Windows 10/64-bit)

1. **Install or activate Bash on Windows using Powershell**:
```
$ Enable-WindowsOptionalFeature -Online -FeatureName
Microsoft-Windows-Subsystem-Linux
```
2. **Open Bash and install the dependencies**
```
$ sudo add-apt-repository ppa:avsm/ppa
$ sudo apt-get update
$ sudo apt-get install ocaml ocaml-native-compilers camlp4-extra opam
$ sudo apt-get install make
$ opam init
$ sudo apt-get install llvm
$ eval `opam config env`
$ wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key | sudo apt-key
add -
$ sudo apt-get update
$ sudo apt-add-repository "deb http://apt.llvm.org/xenial/
llvm-toolchain-xenial-6.0 main"
$ sudo apt-get install -y clang-6.0
$ sudo apt-get install libpng12-0
```

2.2 Using the Compiler

Copy the Colode repository from GitHub: `https://github.com/tfmunat/Colode.git`
It can also be cloned by using SSH: `git@github.com:tfmunat/Colode.git`

1. **Build Compiler:**
```
$ make
```
2. **Run and validate integration tests:**
```
$ ./integration_tests.sh
```
3. **Commands**
```
$ make -> to build
$ make clean -> to clean
```

2.3 Sample Program

```
// helloWorld.colode

print("Hello World!");
```

3 Language Reference Manual

3.1 Types

Types define how data can be stored and in what format, which allows the compiler to know how data will be used. Colode is a strictly typed language, so the type of each variable and return type of every function must be specified. Each type is defined by a variable name, variable type, and an attributed value (either during initialization or after). Names can be alphanumeric, are case sensitive, and may contain "_" to connect two or more words.

3.1.1 Primitive Data Types

Types	**Description**	**Examples**
int	4 byte signed integer type. Overflow not allowed. Values between -2,147,483,648 and 2,147,483,647.	`int x; x = 45;` `int y = 45;`
float	8-byte floating point number.	`float y;` `float x = 47.35;`
bool	8-bit boolean variable represented by true (1) or false (0).	`bool x = true;` `bool y = false;`
char	1-byte ASCII character.	`char x = ‘x’;`

list	Lists are an ordered, iterable collection of single-type data. Indexed using bracket notation.	`char list listName = [ ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ ];`
		`listName.0; // ‘a’`
		`listName.4; // ‘e’`

string	Immutable list of chars.	`string argName = “hello”;`
void	Function returns void when there is nothing to return.	`def void funcName():`

3.1.2 Built-in Data Types

Types	Description	Examples
Image	Images hold pixel values in a struct of 4 width x height matrices (1 for each channel). Images are initialized using the `coload` builtin function. @ is used to access the channel matrices of the image. Properties: - *int* `width` - *int* `height` - *Matrix* `red` - *Matrix* `blue` - *Matrix* `green` - *Matrix* `alpha`	`Image img = coload("omg.jpg");` `img@blue = img@red;`

Matrix	Matrices are a 2 dimensional data structure of fixed row length and column height (these are set implicitly at declaration). Values may be indexed (from 0) by their row and column integer, e.g. `mat.`*`row`*`|`*`column`*	`Matrix m = [ 0 0 1 | 0 1 0 | 1 0 0];`
		`m.0|2 ; // 1`
		`m.1|0 ; // 0`

3.2 Lexical Conventions

3.2.1 Identifiers

Identifiers are used for variable declaration. Identifiers must begin with a lowercase letter, and can then be followed by any order of letters, numbers, and underscores (an identifier that tersely explains the data type is recommended).

3.2.2 Keywords

Keywords are reserved, and thus cannot be used as identifiers, because they have a special meaning to the Colode program. The following table demonstrates the specifications.

`int`	`float`	`bool`	`string`	`false`
`true`	`return`	`continue`	`break`	`else`
`elif`	`if`	`while`	`for`	`def`
`void`	`in`	`range`	`Image`	`Matrix`

3.2.3 Literals

Integer	A sequence of digits	`0, 3, 7162, 10`
Float	A number literal with an integer and fraction part. The integer part is written first, followed by a dot (.) followed by a fractional part.	`0., 3.1415`
Boolean	The keyword `true` refers to a truth value (1), while `false` refers to a false value (0)	`true, false`
Character	A single ASCII character surrounded by single quotes. The ‘ character must be backslash-escaped within character literals.	`‘A’, ‘!’, ‘\t’`
String	A sequence of ASCII characters surrounded by double quotes. The “ character must be backslash-escaped within string literals. Additional escape sequences are as follows: `\n` New line `\r` Carriage return `\t` Tab `\\` Backslash	`“The quick brown fox jumped over the lazy dog”, “Hello World!\n”`
Matrix	Matrix literals are sequences of numeric literals in square braces (the left brace is preceded by a tilde). Each row is delimited by the pipe character. Row items are space-delimited.	`~[0 1 | 0 1], ~[0 0 0], ~[2.5 |0.2]`

3.2.4 Comments

Comments are single-line only, and denoted by a double forward-slash ‘//’

```
// This is a comment
// This is another comment
```

3.2.5 Operators

Operator	**Description**	**Associativity**
Arithmetic	`+, -, *, /, ^`	Left to Right
Comparison	`<, >, <=, >=, ==, !=`	Left to Right
Assignment	`=, +=, -=, *=, /=`	Right to Left

3.2.6 Arithmetic

Arithmetic operators are used on the primitive data type of int and float. The resulting type is preserved. Currently supported operations are addition(+), subtraction(-), multiplication(*), division(/), raising-power(^), and mod(%).

For the additive operators, both operands are qualified versions of compatible object types. The result of the ‘+’ operator is the sum of the operands. The result of the ‘-’ operator is the difference of the operands.

Operands of ‘*’ and ‘/’ must have arithmetic type. The binary ‘*’ operator indicates multiplication, and its result is the product of the operands. The binary ‘/’ operator indicates division of the first operator (dividend) by the second (divisor). If the value of the divisor is 0, it would throw an error. The binary ‘%’ operator yields the remainder from the division of the first expression (dividend) by the second (divisor). The remainder has the same sign as the dividend, so that the following equality is true when the divisor is nonzero: *(dividend / divisor) * divisor* + *dividend % divisor* == *dividend*. If the value of the divisor is 0, it would throw and error.

Expression	**Result**	**Comments**
`3 + 2 * 6`	`15`	Multiplication is done before addition.
`3 + (2 * 6)`	`15`	Parentheses follow the precedence rules
`(3 + 2) * 6`	`30`	Parentheses override the precedence rules.

3.2.8 Comparison

The relational(`<  <=  >  >=`) and equality(`==  !=`) comparison operators are of binary class with a left to right associativity. The operators <(less than), >(greater than), <=(less than or equal to), and >=(greater than or equal to) all yield a result of type int with the value 0 if the specified relation is false, and 1 if it is true.

The operands must be both arithmetic, in which case the usual arithmetic conversions are performed on them.

Expression	Result	Comments
`4 + 1 > 2`	`1 (true)`	Addition is done before comparison
`(4 + 1) >= 2`	`1 (true)`	Parentheses follow the precedence rules

The ==(equal to) and the != (not equal to) operators are exactly analogous to the relational operators, except for their lower precedence. For example, a < b == c < d is 1 whenever a < b and c < d have the same truth value.

Expression	Result	Comments
`4 + 1 == 2 + 3`	`1 (true)`	Addition is done before equality comparison
`(4 + 1) == 2 + 3`	`1 (true)`	Parentheses follow the precedence rules

3.2.9 Assignment

All assignment operators associate from right to left. The operands permissible in simple assignment(=) must be both arithmetic types or are of compatible structures. In the case of ‘=’ the value of the right operand is converted to the type of the assignment expression, and replaces the value of the object referred to by the left operand.

For the operators += and -=, both operators must have arithmetic types.

3.2.10 Matrix operations

Colode includes support for a number of matrix operations. The arithmetic operations work on matrices (per-element scalar operations), but convolution is also supported.

Convolution	**	A binary operator that returns the two-dimensional convolution of its two operand matrices.	`Matrix a = [1 1 1|0 0 0|1 0 1];`
			`Matrix b = a;`
			`Matrix c = a ** b;`

3.2.11 Punctuation

Each statement (besides statement blocks) are punctuated with a semicolon at the end to denote sequencing. Array literals use commas to separate items, and function calls use commas to separate function arguments. A pair of expressions separated by a comma is evaluated left to right.

```
int a = random(seed, param);
bool list b = [ true, false ];
```

3.3 Syntax Notation

Colode programs are a series of statements and declarations that are executed in top-to-bottom order.

3.3.1 Expressions

An expression is a combination of values, variables, operators, and functions to be evaluated. There are two types of expressions: one is assignment, and the other is typically arithmetic or boolean expressions.

3.3.2 Assignment

<identifier> [= <expression>];
Expression is evaluated and stored into the identifier.

Example:
`x = 10`
is an assignment expression which evaluates to 10.

3.3.3 Arithmetic Expressions

<expr_1> <op> *<expr_2>*;

Example:
`2 + 3`
is an arithmetic expression, which is composed of literals (2 and 3) and an arithmetic operator +. This evaluates to 5.

3.3.4 Boolean Expressions

```
<expr evaluates to true or false>;
```

Example:
`2 != 5`
is a boolean expression, which is composed of literals (2 and 5) and a comparison operator !=. This evaluates to true.

3.4 Declarations

3.4.1 Variable Declaration

The syntax for declaring variables is:

```
<type> <identifier> [= <initial_val>];
```

Variables may be optionally initialized at their declaration. To declare the variable as a list type, the individual type must be written together with list, e.g. int list a;

```
int i;
bool b;
string s = “Hello World”;
string list buf = [“The”, “quick”, “brown”, “fox”]
```

3.4.2 Function Declaration

The syntax for function declaration is:

```
def <identifier> ([<type> <param>, ...]) : <return type> {
      <function body>
}
```

Functions may only be declared at the top level of a Colode file, not within another function. Functions that do not have a void return type must have a return statement that corresponds to the return type in the function declaration.

3.5 Initialization

Variables may be initialized at or after their declaration, by assigning an expression of corresponding type

```
int i;
i = 7;
```

```
int n = 16;
string s = "Hello World";
```

3.6 Statements

A statement may be either an expression, a variable declaration, or one of the following control-flow statements. All statements that do not contain blocks must be ended with a semicolon.

3.6.1 if / elif / else

"if" is used for executing conditional statements. The syntax for an if statement is:

```
if <expr> {
      <suite>;
}
```

expr must be a boolean expression to be evaluated, and suite is a sequence of statements. If expr is true, then the program goes into the brackets and executes suite. If expr is false, then the program ignores suite and proceeds to the next operation.

"if / elif / else" is used for multiple conditional branching; the syntax is:

```
if <expr_1> {
      <suite_1>;
}
elif <expr_2>{
      <suite_2>;
}
elif <expr_3>{
      <suite_3>;
}
......
else {
      <suite_n>;
}
```

The program sees each boolean expression one by one and executes a suite only when the corresponding expression is true. If nothing matches, then the suite in the else statement is executed. Boolean expressions, `expr_1` through `expr_n` *should be mutually exclusive* to ensure a logically correct control flow. If not, the programmer has to make sure that earlier expressions are caught in the order they are expected. See Example 2 below regarding this comment.

Example:

```
int i = 5;
if i%3 == 1{
      print(“i = 3k+1”);
}
elif i%3 == 2{
      print(“i = 3k+2”);
}
else {
      print(“i = 3k”);
}
```

Example 2: ** *Notice how the first 2 expressions are both true! So the programmer has to specify the order of the expressions carefully. The control flow here is just like Java* **

```
int i = 6;
if i%2 == 1{
      print(“i = 3k+1”);
}
elif i%3 == 2 {
      print(“i = 3k+2”);
}
else {
      print(“i = 3k”);
}
```

3.6.2 Loops (for, while, continue, break)

“for” is used for iterative executions with an update statement. The syntax is as follows:

```
for <statement>; <statement>; <statement>; {
      <statement>;
}
```

The first statement is executed before the loop is run. The second is the condition that is checked at the beginning of each loop. If the second statement evaluates to false, the loop breaks, or else it continues. The third statement in the declaration is the update statement, which is run at the end of each loop

Example 1:

```
for int x = 2; x <= 7; x = x + 2{
      iprint(x);
}     //print “2 3 5 7” on screen
```

“while” is also used for iterative execution. An iteration continues as long as the specified condition is true. The syntax is:

```
while <condition> {
      <statement>;
}
```

condition must be a boolean expression.

Example:

```
int i = 5;
while i > 0 {
      print(“Hello”);
      i = i - 1;;
}     //print “Hello” five times on screen
```

“continue” is used for ending current iteration of a for/while loop and starting with the next iteration, followed by a semicolon.

“break” is used for ending the iteration of the nearest enclosing for/while loop, followed by a semicolon.

3.6.3 Return

Return is used for ending the current function execution and return a value or multiples values

```
def times(int a, int b) : int {
      return a * b;
}
```

3.6.4 Statement Blocks

Statement blocks refer to a list of statements surrounded by curly braces that follow if/elif/else, for, while, as well as function declarations.

3.7 Standard Library Functions

The following are the standard library functions for the Colode programming language.

The Colode standard library provides functions for tasks such as image cloning, rotating, cropping, zooming etc. Currently, we do not handle memory management.

Function	Description	Usage
coload	Load any source image file for another available operation. Must be a valid image file of type .jpg or .png. Takes the image name(with the file extension) as the only parameter. Images must be in the same working directory.	`Image img = coload("image.png");`
coclose	Close a source image file. Takes an already initialized “Image”, and the name of file to save.	`coclose(img, “modified.png”);`
print	Takes a String as the parameter and prints it to the terminal.	`print(“Hello World!”);`
iprint	Takes an Int as the parameter and prints it to the terminal.	`print(10);`
fprint	Takes a Float as the parameter and prints it to the terminal.	`print(10.1);`
mprint	Takes a Matrix as the parameter and prints it to the terminal.	`mprint(img@blue * img@red);`
new	Takes two integers (width, height) and creates an empty zeroed-out Matrix with those dimensions	`new(i, j);`
height	Takes a matrix and returns it height	`height(mat);`
width	Takes a matrix and returns its width	`width(mat);`
generate_gaussian	Takes two integers to specify size, and a float value to specify sigma. Returns a gaussian kernel matrix.	`generate_gaussian(5, 5, 10.0);`

generate_brighten	Takes a float to specify intensity (on a scale of 1) and returns a brightening kernel matrix.	`generate_brighten(1.5);`
generate_sharpen	Returns the default sharpening kernel matrix.	`generate_sharpen();`
generate_edge_detect	Returns the default edge detect matrix.	`generate_edge_detect()`

4 Project Plan

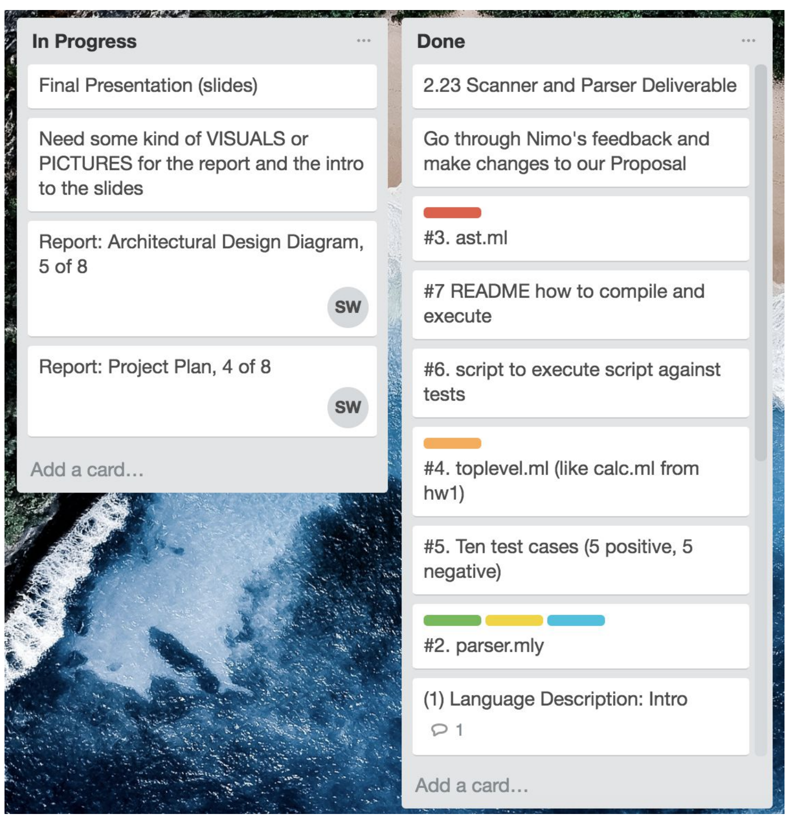
4.1 Planning, Specification, and Development

Our three primary tools were GitHub, Slack, and Trello. The first allowed us to have a collective interface for our code, the second gave a comfortable manner for us to communicate effectively as a team, and the latter enabled us to split the work evenly throughout the group.

4.1.1 Dividing up Tasks

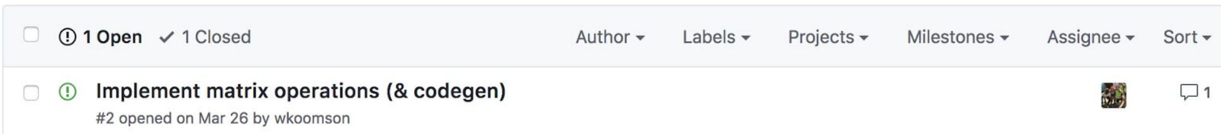
Trello was initially suggested by the Manager. After a discussion and approval of each member to trying out Trello, the group would have a meeting to discuss how to divide up the work appropriately and the backlog was populated appropriately.

Then the Manager would hand out tasks and spread the work evenly, assigning a color to each person's card or task on the backlog for convenience and ease of use. This is an agile management tool that allowed every member to see what needed to be accomplished, how long it would take, and the unpin it once it was done. Below is a sample of our trello usage:



4.1.2 Dealing with Issues

Using GitHub, we were able to create log records of what the current issue was. Here is a sample image showing this process:



4.2 Timeline

4.2.1 Milestone

Date	Milestone
Feb. 2	Project Proposal
Feb. 21	Scanner and Parser
Feb. 26	Language Reference Manual
Mar. 12	Control Flow structures (If-else, for/while loop)
Mar. 26	Hello World
Apr. 4	Matrices completed
Apr. 18	Extended Testsuite
Apr. 22	Demo Version 1 complete
Apr. 24	Matrix convolution complete
Apr. 28	Demo Version 2 complete
May 8	Presentation complete
May 9	Final report complete

4.2.2 GitHub

Below are representation to show our work progress. Figures taken from GitHub:

4.3 Roles and Responsibilities

Members	UNI	Role	Responsbility
Steven Bonilla	sb3914	Manager	Higher level operations such as organizing group meetings, delegating tasks, and debugging when necessary. Writing documentation for the code.
Tahmid Munat	tfm2109	Language Guru	Pushing the group to implement as many of the features indicated on the LRM. Language design, scanner and sast implementation. Debugging and writing documentation.
Takuma Yasuda	ty2353	Language Guru	Implemented a top-level, and wrote documentation for the code.
Willie Koomson	wvk2101	System Architect	Implementing lower-level operations such as codegen and managing the software stack for Colode.
Dimitri Borgers	djb2195	Tester	Creating and maintaining a set of tests, as well as ensuring new changes are in compliance with the test suite.

4.4 Project Log

```
commit 79c5d9c488f87dea6fa39a936558a70a2b6759d0
Date:   Wed May 9 19:09:37 2018 -0400

        Merge branch 'changes' of
https://github.com/tfmunat/Colode

commit 9fab2f44b5f544eff3093c494c954e63375f8e14
Merge: c5557eb bc71bc8
Date:   Mon May 7 20:31:04 2018 -0400

    Merge branch 'changes' of
https://github.com/tfmunat/Colode into changes

commit c5557eb872be44e00d9e892173b37b746aeec52a
Date:   Mon May 7 20:27:57 2018 -0400

    added some things for presentation

commit bc71bc8bc4f6ba62ce72eea40949c3a828efb6ae
Date:   Mon May 7 20:00:10 2018 -0400

    minor folder restructuring

commit 3c01d5555a32dcdc274e57e32ee7c2efabe894fe
Date:   Sun May 6 15:11:17 2018 -0400

    Working convolution !!!

commit 11b8453672cc6e61740b8b4a3e15950b00fa947a
Date:   Sun May 6 12:11:35 2018 -0400

    png read and write functions working

commit d9fc9a3ff73e6587b8f2677c45793128a217c0a4
Date:   Fri May 4 23:07:04 2018 -0400

    more matrix funcs

commit 61c80d728c568365f97646ee89333aa30ad22b57
Date:   Fri May 4 22:05:43 2018 -0400
```

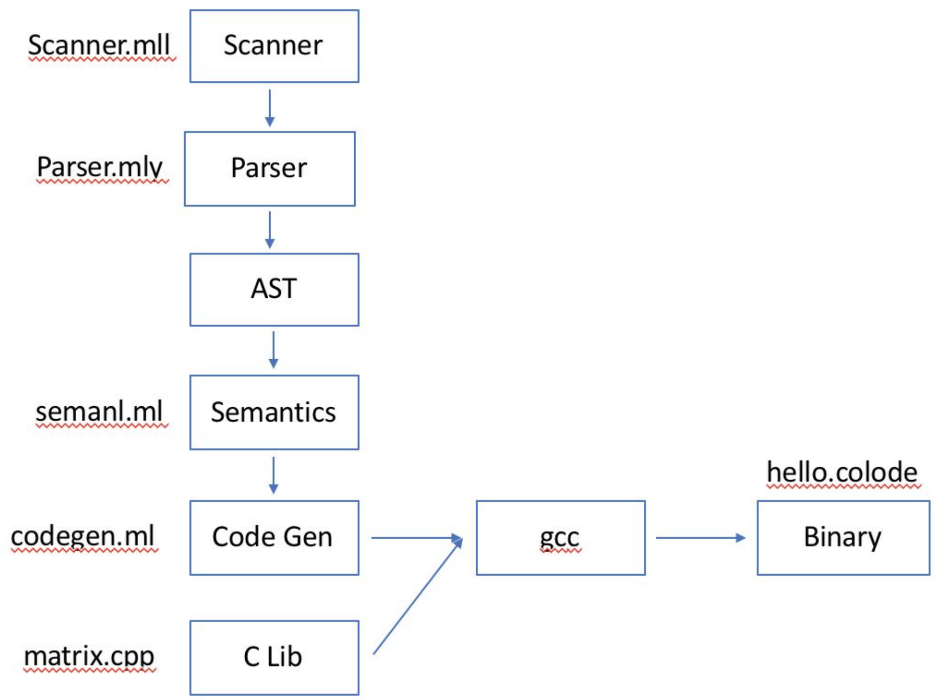
```
    some matrix functions working

commit 031802029e2ec5cfb1884c45da9bb23fdff422c3
Date:   Wed Apr 18 18:12:56 2018 -0400

    added LRM to misc

commit 65550dd50e21166cb3420988e28b20176cb80600
Date:   Wed Apr 18 18:12:06 2018 -0400

    update gitignore

commit 1d4b5bedf4f79f5882863ab5ac633a6d9bfe24bb
Date:   Wed Apr 18 18:10:49 2018 -0400

    added empty file that prevents some
ocamlbuild warning messages about recursive
compilation

commit 910804c6e44a9bc7297f4c4b6fcf5ac1243e6e68
Date:   Wed Apr 18 18:09:24 2018 -0400

    moving things around

commit 38e18debc43c39a70ed6f82ce67fa89383c0fa73
Date:   Wed Apr 18 18:08:28 2018 -0400

    integration tests

commit c757d37473e63df99957f893e15ca78b925beebc
Date:   Wed Apr 18 18:07:04 2018 -0400

    things working well

commit 8875772ba57315a50aed0bf10057eaf88b5a8e29
Date:   Tue Apr 17 12:19:28 2018 -0400

    string concat test

commit a129d1eecd36663a986aba3dbdf2a00918930586
Date:   Tue Apr 17 12:18:54 2018 -0400

    string concat working414

commit 73a6e35f0cfeee4d912d466b7db193a8258e42c4
Date:   Tue Apr 17 11:57:53 2018 -0400

    string concatenation almost working

commit 1de8a3a58360acd241d4986265faf801f507d656
Date:   Sun Apr 15 17:19:28 2018 -0400

    added for and if

commit 49723cfc19fb25b460df696cd2f9ed0774ccbec8
Date:   Sun Apr 15 16:39:55 2018 -0400

    More of codegen

commit 44a63b757d968f8799555a2af19387d640cbc4d6
Date:   Wed Apr 11 19:20:56 2018 -0400

    codegen for arrays, indexes and assignments

commit d5ee78cb1590e7662f6426786387c9eecefedadf
Date:   Wed Apr 11 19:18:47 2018 -0400

    fixed lexing of string literals

commit 0557ced240da5b23724cfa1935c86c1c3f0f9aea
Date:   Wed Apr 11 19:18:27 2018 -0400

    moved old tests

commit 0ab2ddb45149376a7b3899f943005226a940c3f7
Date:   Mon Mar 26 18:54:00 2018 -0400

    README

commit 93480d9af80ada3541fb2c6e5ea3e67c3d37b18d
Date:   Mon Mar 26 18:52:28 2018 -0400

    Updated README

commit 22aa1b3fac16e568b2da787b4a6f9e91d2818d2f
Date:   Mon Mar 26 18:51:07 2018 -0400

    updated compiler instructions

commit 9421573804eae737b2cb09117752f6688205891b
Date:   Mon Mar 26 18:41:47 2018 -0400

    commented out unused

commit f4ab6ae11303377be36b249a135ed3c6be81d946
Date:   Mon Mar 26 18:37:20 2018 -0400

    commented out array index grammars

commit cc928bf783005f4d9bdc91944b088cea21593a6f
Merge: 0093cf0 aea1a9e
Date:   Mon Mar 26 18:32:08 2018 -0400

    Merge branch 'master' of
https://github.com/tfmunat/Colode

commit 0093cf0a00ec2823006c171729b31c268bda180a
Date:   Mon Mar 26 18:31:57 2018 -0400

    fixed reduce/reduce

commit aea1a9e87e074b16e0e65a5d005ecbb49599069c
Date:   Mon Mar 26 18:23:52 2018 -0400

    Update README.md

commit 71eac3a63ef8aeb26b475dca95f5f4f1d85c1b54
Date:   Mon Mar 26 18:20:56 2018 -0400

    Update README.md

commit 5f7f784dfa170d598c52fb4de915d655f7e6a4eb
Merge: 0990a71 0995ed1
Date:   Mon Mar 26 18:18:48 2018 -0400

    Merge branch 'master' of
https://github.com/tfmunat/Colode

commit 0990a71f3ca6fc83e4bc0a9dd168f6aac22da9a5
Date:   Mon Mar 26 18:16:23 2018 -0400
```

```
    fixed extra quotes

commit 0995ed187747e37327d2d33e5b968c0a41c14fa5
Date:   Mon Mar 26 18:16:05 2018 -0400

    Update and rename README to README.md

commit 2aa7ce07b7c5155d911cd24c090c6a33e1f32269
Date:   Mon Mar 26 18:04:53 2018 -0400

    folder restructuring

commit a472583c363170ce19a573285c03ca1a869e1819
Date:   Mon Mar 26 17:54:55 2018 -0400

    fixed makefile, moved scripts and files

commit 072da6c415c9fee8a802d9f9b26eca372ca29c5b
Date:   Mon Mar 26 17:34:46 2018 -0400

    simple readme

commit d1a05e1e2c877397d9e2cf2e3b290d767ac1acae
Date:   Mon Mar 26 17:29:34 2018 -0400

    add 'integration' test

commit 46384abaa1c0135c591dd591c94af7166ecd0ef4
Date:   Mon Mar 26 17:05:42 2018 -0400

    codegen working

commit 5094c494016836afdec4be7192d9252a83fbdf5b
Date:   Mon Mar 26 15:22:48 2018 -0400

    current codegen

commit b8fa6ecc5086d5c25630626e047dbe3b58d4cbd2
Date:   Mon Mar 26 12:52:56 2018 -0400

    codegen working for print()

commit d6e30f3fd60f1cc4f54e6a4b10a505addcedb514
Date:   Sun Mar 25 20:39:20 2018 -0400

    codegen

commit 9d1386a65b8c40a252a72720b30fdb7f19d4c15c
Date:   Sun Mar 25 19:55:04 2018 -0400

    started on codegen

commit 0280cbf11f06765f700df966817ae50416d26d20
Date:   Sat Mar 24 23:40:15 2018 -0400

    semantic checker compiles

commit 4ebe6a65c73129f3c8af331c1e6ae8cde7985781
Date:   Sat Mar 24 22:21:23 2018 -0400

    almost done with semantic

commit bd90621749f40dd545516cee46565ac22d6672db
Date:   Sat Mar 24 18:53:54 2018 -0400

    almost done with semantic

commit 41753115750bf8c0e973febf584357db88fc208b
Date:   Fri Mar 23 23:20:41 2018 -0400

    started sematic checker

commit 4a3461553c64c992dd0ef63bea43ee368c711e0e
Date:   Fri Mar 23 21:57:37 2018 -0400

    added sast types

commit 9c6549391f977bc53c07d9e88be858cd61537028
Date:   Wed Feb 21 23:38:13 2018 -0500

    updated instructions for resubmission

commit ce56319cb65a56c9939089f56903ab0d7fcfd42c
Date:   Wed Feb 21 23:30:10 2018 -0500

    brushed up scanner and parser string lexing;
conflicts still present

commit 57b0f1ae4a21b919f9c77562a3f4e4457ac47b6c
Date:   Wed Feb 21 22:52:54 2018 -0500

    minor adjustments; wont break previous builds

commit 946dc8775fc03e0ff0537b178cec00a95fd2f629
Date:   Wed Feb 21 22:22:55 2018 -0500

    removing bash dependency for tester

commit 9fd5df6cfa95e05792a52016bb73b842f3cc9e51
Date:   Wed Feb 21 22:16:53 2018 -0500

    added a Makefile

commit 5bb96a7e5396a17db45c4a87f092aed9f4f18baa
Merge: 9eacd26 d7bf75b
Date:   Wed Feb 21 22:01:29 2018 -0500

    Merge branch 'master' of
https://github.com/tfmunat/Colode

commit 9eacd26d9995dc57614f09a29fc17053847a9882
Date:   Wed Feb 21 22:01:14 2018 -0500

    Delete unecessary files

commit d7bf75b00c392a500aa32930e38ac859ce4891a6
Date:   Wed Feb 21 21:58:35 2018 -0500

    README

commit 90cb31f40b47e3dfedaf7fd014c6d49c6d79b6dd
Date:   Wed Feb 21 21:56:09 2018 -0500

    added exponent to parser and tests running

commit b525464960f934ecf3a1e7ee076e22fa831b6e8a
Date:   Wed Feb 21 21:48:43 2018 -0500

    added some more tests

commit 0c41cf890bb5c23c304586cabe6b3115fa1e89e5
Date:   Wed Feb 21 18:42:28 2018 -0500
```

```
    committed sample file

commit 79d6be8726575c8407c63a6e2d8ae795a156331c
Merge: b66e53a bd7893f
Date:   Wed Feb 21 18:41:21 2018 -0500

    Merge branch 'master' of
https://github.com/tfmunat/Colode

commit b66e53ae29401d2b9ce7edf03e6054b4c6b7c0c7
Date:   Wed Feb 21 18:41:14 2018 -0500

    added array index, member access (dot
operator), and changed to semi colon sequence

commit bd7893f2312dabc97170cf855465bcfaa53b30f9
Date:   Wed Feb 21 17:56:38 2018 -0500

    cleaning up

commit 3cc478a0bccd24eb4e13575df7cbdb1d08ee3762
Date:   Wed Feb 21 17:52:17 2018 -0500

    tester, clean, build bash

commit 1b2ead77ec07ce42c8b2b67e9e5cfce350f8a8a0
Merge: d6d94a3 767aaa0
Date:   Wed Feb 21 17:51:22 2018 -0500

    Merge branch 'master' of
https://github.com/tfmunat/Colode

commit 767aaa00881434a1d1a4559b4247cb86d60150b2
Date:   Wed Feb 21 16:40:45 2018 -0500

    fixed shift/reduce conflicts

commit d6d94a3f7799ede3869638d6fd1be20a1c306891
Date:   Wed Feb 21 15:49:59 2018 -0500

    added build,clean,tester

commit aa1e2cda7039d452151462bb5e0a41599874827c
Date:   Tue Feb 20 19:35:20 2018 -0500

    added char and string literals

commit dd70edd55a19ea3657631aaaa4d7c88c3aa38de7
Date:   Tue Feb 20 19:10:08 2018 -0500

    fixed func declarations & added array literal
parsing

commit 10efdc31fbc5e57a499c7207286ac2061a562d80
Merge: 2aacc94 ec6cd85
Date:   Tue Feb 20 14:11:37 2018 -0500

    Merge branch 'master' of
https://github.com/tfmunat/Colode

commit 2aacc94a902d130628419b02df442c93177d2662
Date:   Tue Feb 20 14:04:36 2018 -0500

    restricted top-level functionality to
scanning and parsing just for this time

commit ec6cd85480c1340d51c6c5ae900bf47476ee235c
Date:   Tue Feb 20 10:43:13 2018 -0500

    README update

commit 567976b4a5c4c3b99145ad56ca3626eed0a54c30
Date:   Tue Feb 20 10:39:35 2018 -0500

    Added build instructions

commit c85a726e7b8764bc2f010f262be6acb529421388
Date:   Tue Feb 20 10:10:55 2018 -0500

    first write of toplevel

commit 0b88f322fe322d088d15b06d925ccdaa6fcbcc87
Date:   Mon Feb 19 21:46:31 2018 -0500

    removed compile artifact from repo

commit 4120dc1fb254fcb3f00543b35bb5a6cb9f0a015f
Date:   Mon Feb 19 21:45:59 2018 -0500

    fixing compile errors

commit 23963b963a20ce1e284c8e1bc9194be1e7753ec8
Date:   Mon Feb 19 21:25:19 2018 -0500

    parser mostly done?

commit aced4d17b75b4c5e65a7911faddcc88a2e8499f6
Date:   Wed Feb 14 20:08:17 2018 -0500

    added our tokens to parser & TODOs

commit aeb77f0add0c6f33aefc84777923a99693424b1f
Date:   Wed Feb 14 19:49:56 2018 -0500

    Added convolution operator, curly braces

commit 28e0392a0bf3561f73af8eeb9fab8389bd9cdb88
Date:   Fri Feb 9 17:30:04 2018 -0500

    added our tokens to the scanner

commit e1b31137cfffe198e22bf116ab1ecdc580f96de0
Date:   Fri Feb 9 16:28:49 2018 -0500

    added empty files for Syntax assignment


    Upload proposal

commit 3b07981ca7ce65b60004096abdcd2a6cde11e5a4
Date:   Wed Jan 31 20:35:27 2018 -0500

    Initial commit
```

5 Architectural Design

5.1 Diagram



5.2 Compiler

Our compiler is implemented in a number of passes. The scanner transforms the input into a series of tokens, which are then parsed into an abstract syntax tree by the parser module. The AST is checked for correct semantics in the semantic checker module, and then the codegen module takes a semantically-checked syntax tree and generates the corresponding LLVM IR. The IR is then compiled using llc, and linked with the matrix and image functions using gcc/ld.

5.2.1 Scanner.mll

The scanner module reads the file specified on the compiler command, and then generates a stream of tokens corresponding to language structure, keywords and identifiers. Scanner may throw error if it encounters characters that are not valid tokens in the language.
Implemented by: Tahmid Munat

5.2.2 Parser.mly

The parser module is implemented as a set of Yacc rules that read a token stream into an abstract syntax tree, and ensures that they are syntactically correct. The precedence and association rules of all Colode operators are defined here.
Implemented by: Dimitri Borgers

5.2.3 AST

The abstract syntax tree representation of the Colode program. Each node of the tree denotes a construct occurring in the source code. This representation is created after tokens have been run through the parser.
Implemented by: Steven Bonilla

5.2.4 Semant.ml

The semantic checker module traverses the entire AST in a depth-first manner, checking each expression along the way for violations of Colode's semantic and type rules. The output of this module is a semantically-checked AST (SAST).
Implemented by: Willie Koomson

5.2.5 Codegen.ml

Takes in the SAST created by the Semant.ml file and outputs LLVM IR bytecode.
Implemented by: Willie Koomson

5.2.6 Matrix.c

Matrix.c contains the C implementations of our matrix operations, such as matrix multiplication and convolution. These are compiled into an object and linked into the final Colode executable, `hello.colode`.
Implemented by: Willie Koomson

6 Test Plan

6.1 Test Scripts

We used a Bash script to drive our compiler on our tests, and evaluate the result.
Implemented by: Dimitri/contributions, evaluations, and revisions from each teammate.

6.1.1 Integration_tests.sh

```
compiler="./toplevel.native"
make clean
make
if [ $? -ne 0 ]; then
       echo "Compilation failed."
       exit 1
fi
success=0
failure=0
for t in ./Tests/*.cld;
       do
              echo "================="
              name=$(head -1 $t | sed 's/^\/\///')
              echo $name
              echo "-----------------"
              $compiler -l $t > test.ll
              res=$?
              if echo $t | grep -qi "fail" ; then
                     if [ $res -eq 0 ]; then
                            echo "$name failed. Compilation should have failed."
                            failure=$((failure +1))
                     else
                            echo "$name succeeded. Compilation failed correctly."
                            success=$((success +1))
                     fi
              else
                     expected=$(head -2 $t | tail -1 | sed 's/^\/\///')
                     llc test.ll > test.s
                     gcc -fPIC -lc -static-libgcc -lm -no-pie -o test test.s matrix.o
                     output=$(./test)
                     echo $output
                     echo $output | grep -qa "$expected"
                     if [ $? -eq 0 ]; then
                            echo "$name succeeded; output matched expected: $expected"
                            success=$((success +1))
                     else
```

```
                          echo "$name failed; output did not match expected:
$expected"
                          failure=$((failure +1))
                    fi
             fi
             rm -f test.ll test.s
             echo ""
      done
echo "================="
echo "$success tests passed and $failure tests failure."
```

6.1.2 Arrayconcat.cld

```
//Array concatenation test
//4
int list a = [1,2];
int list b = [3,4];
int list c = a + b;
iprint(c.3);
```

6.1.3 Assign_fail.cld

```
// Invalid assignment test
string blue = 1;
```

6.1.4 Assignadd.cld

```
//Assign Add test
//2
int one = 1;
one+=1;
iprint(one);
```

6.1.5 Assignadd_fail.cld

```
//AssignAdd w/ wrong type test
int one = 1;
one+= 1.0 ;
iprint(one);
```

6.1.6 Declassignment.cld

```
//DeclAssignment expression test
//30
int a = int b = 30;
iprint(a);
```

6.1.7 Exponentiation.cld

```
//Exponentiation Test
//4.0
float a = 2.0;
a = a^2.0;
fprint(a);
```

6.1.8 If.cld

```
//If test
//Yes
int a = 3;
int b = 2 + 1;
if a == b {
	print("Yes");
} else {
	print("No");
}
```

6.1.9 Stringcompare.cld

```
//String Comparison Test
//Yes
string a = "ABC";
string b = "ABC";
if b == a {
	print("Yes");
} else {
	print("No");
}
```

6.1.10 Stringconcat.cld

```
//String Concatenation test
//Hello World
string greeting = "Hello" ;
string g = greeting + " World";
print(g);
```

6.1.11 While.cld

```
//While test
//2
int a = 0;
```

```
while a != 2 {
	a += 1;
}

iprint(a);
```

6.1.12 While_fail.cld

```
// While w/ invalid predicate
int three = 3;

while three {
	print("Yaasss");
}
```

6.2 Example Programs

The programs below showcase the standard library functions of Colode as well as more its algorithmic abilities.

6.2.1 Program 1

```
Image img = coload("richard.png");
Image img2 = coload("richard.png");
Image img3 = coload("richard.png");
Image img4 = coload("richard.png");
Image img5 = coload("richard.png");
matrix gaussian = generate_gaussian(5, 5, 10.0);
print("Gaussian");
mprint(gaussian);
print("Center element");
fprint(gaussian.2|2);
img@red = img@red ** gaussian;
img@blue = img@blue ** gaussian;
img@green = img@green ** gaussian;
coclose(img, "richard_gaussian.png");

print("Sharpen");
matrix sharpen = generate_sharpen();
mprint(sharpen);
img2@red = img2@red ** sharpen;
img2@blue = img2@blue ** sharpen;
img2@green = img2@green ** sharpen;
```

```
coclose(img2, "richard_sharpen.png");

print("Edge detect");
matrix edge = generate_edge_detect();
mprint(edge);
img3@red = img3@red ** sharpen ** edge;
img3@blue = img3@blue ** sharpen ** edge;
img3@green = img3@green ** sharpen ** edge;
coclose(img3, "richard_edge_detect.png");

matrix brighten = generate_brighten(1.5);
mprint(brighten);
img4@red = img4@red ** brighten ** gaussian;
img4@blue = img4@blue ** brighten ** gaussian;
img4@green = img4@green ** brighten ** gaussian;
coclose(img3, "richard_bright_gaussian.png");

img5@blue = img5@green = img5@red;
coclose(img5, "richard_grayscale.png");
```

6.2.2 LLVM Output for Program 1

```
; ModuleID = 'Colode'
source_filename = "Colode"

@0 = private unnamed_addr constant [12 x i8] c"richard.png\00"
@1 = private unnamed_addr constant [12 x i8] c"richard.png\00"
@2 = private unnamed_addr constant [12 x i8] c"richard.png\00"
@3 = private unnamed_addr constant [12 x i8] c"richard.png\00"
@4 = private unnamed_addr constant [12 x i8] c"richard.png\00"
@5 = private unnamed_addr constant [9 x i8] c"Gaussian\00"
@6 = private unnamed_addr constant [15 x i8] c"Center element\00"
@7 = private unnamed_addr constant [3 x i8] c"%f\00"
@8 = private unnamed_addr constant [21 x i8] c"richard_gaussian.png\00"
@9 = private unnamed_addr constant [8 x i8] c"Sharpen\00"
@10 = private unnamed_addr constant [20 x i8] c"richard_sharpen.png\00"
@11 = private unnamed_addr constant [12 x i8] c"Edge detect\00"
@12 = private unnamed_addr constant [24 x i8] c"richard_edge_detect.png\00"
@13 = private unnamed_addr constant [28 x i8] c"richard_bright_gaussian.png\00"
@14 = private unnamed_addr constant [22 x i8] c"richard_grayscale.png\00"

declare i32 @puts(i8*)

declare double @pow(double, double)

declare i32 @printf(i8*, ...)
```

```
declare void @_mat_zero_out({ double*, i32, i32 }*)

declare void @_mat_print({ double*, i32, i32 }*)

declare void @_mat_scalar_add({ double*, i32, i32 }*, double, { double*, i32, i32 }*)

declare void @_mat_scalar_subtract({ double*, i32, i32 }*, double, { double*, i32,
i32 }*)

declare void @_mat_scalar_multiply({ double*, i32, i32 }*, double, { double*, i32,
i32 }*)

declare void @_mat_scalar_divide({ double*, i32, i32 }*, double, { double*, i32, i32
}*)

declare void @_mat_mat_add({ double*, i32, i32 }*, { double*, i32, i32 }*, { double*,
i32, i32 }*)

declare void @_mat_mat_subtract({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare void @_mat_mat_multiply({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare void @_mat_mat_divide({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare void @_mat_mat_convolute({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare i1 @_mat_mat_equal({ double*, i32, i32 }*, { double*, i32, i32 }*)

declare void @_mat_mat_power({ double*, i32, i32 }*, i32, { double*, i32, i32 }*)

declare void @_image_read(i8*, { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }*)

declare void @_image_write(i8*, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }*)

declare void @_mat_gen_gauss(double, { double*, i32, i32 }*)

declare void @_mat_gen_sharpen({ double*, i32, i32 }*)

declare void @_mat_gen_edge_detect({ double*, i32, i32 }*)

declare void @_mat_gen_brighten(double, { double*, i32, i32 }*)

define i32 @main() {
entry:
  %img = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  %0 = alloca { i8*, i32 }
  %1 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 0
```

```
  %2 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 1
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @0, i32 0, i32 0), i8** %1
  store i32 11, i32* %2
  %3 = load { i8*, i32 }, { i8*, i32 }* %0
  %4 = extractvalue { i8*, i32 } %3, 0
  %5 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  call void @_image_read(i8* %4, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %5)
  %6 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %5
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %6, { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }* %img
  %img2 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  %7 = alloca { i8*, i32 }
  %8 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %7, i32 0, i32 0
  %9 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %7, i32 0, i32 1
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @1, i32 0, i32 0), i8** %8
  store i32 11, i32* %9
  %10 = load { i8*, i32 }, { i8*, i32 }* %7
  %11 = extractvalue { i8*, i32 } %10, 0
  %12 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  call void @_image_read(i8* %11, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %12)
  %13 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %12
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %13, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2
  %img3 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  %14 = alloca { i8*, i32 }
  %15 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %14, i32 0, i32 0
  %16 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %14, i32 0, i32 1
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @2, i32 0, i32 0), i8** %15
  store i32 11, i32* %16
  %17 = load { i8*, i32 }, { i8*, i32 }* %14
  %18 = extractvalue { i8*, i32 } %17, 0
  %19 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  call void @_image_read(i8* %18, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %19)
  %20 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %19
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %20, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3
```

```
  %img4 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  %21 = alloca { i8*, i32 }
  %22 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %21, i32 0, i32 0
  %23 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %21, i32 0, i32 1
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @3, i32 0, i32 0), i8** %22
  store i32 11, i32* %23
  %24 = load { i8*, i32 }, { i8*, i32 }* %21
  %25 = extractvalue { i8*, i32 } %24, 0
  %26 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  call void @_image_read(i8* %25, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %26)
  %27 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %26
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %27, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4
  %img5 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  %28 = alloca { i8*, i32 }
  %29 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %28, i32 0, i32 0
  %30 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %28, i32 0, i32 1
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @4, i32 0, i32 0), i8** %29
  store i32 11, i32* %30
  %31 = load { i8*, i32 }, { i8*, i32 }* %28
  %32 = extractvalue { i8*, i32 } %31, 0
  %33 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  call void @_image_read(i8* %32, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %33)
  %34 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %33
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %34, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img5
  %gaussian = alloca { double*, i32, i32 }
  %35 = alloca { double*, i32, i32 }
  %36 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %35, i32
0, i32 0
  %37 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %35, i32
0, i32 1
  %38 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %35, i32
0, i32 2
  %39 = alloca double, i32 25
  store double* %39, double** %36
  store i32 5, i32* %37
  store i32 5, i32* %38
  call void @_mat_gen_gauss(double 1.000000e+01, { double*, i32, i32 }* %35)
  %40 = load { double*, i32, i32 }, { double*, i32, i32 }* %35
  store { double*, i32, i32 } %40, { double*, i32, i32 }* %gaussian
  %41 = alloca { i8*, i32 }
```

```
  %42 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %41, i32 0, i32 0
  %43 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %41, i32 0, i32 1
  store i8* getelementptr inbounds ([9 x i8], [9 x i8]* @5, i32 0, i32 0), i8** %42
  store i32 8, i32* %43
  %44 = load { i8*, i32 }, { i8*, i32 }* %41
  %45 = extractvalue { i8*, i32 } %44, 0
  %46 = call i32 @puts(i8* %45)
  %gaussian1 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %47 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian1, { double*, i32, i32 }* %47
  call void @_mat_print({ double*, i32, i32 }* %47)
  %48 = alloca { i8*, i32 }
  %49 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %48, i32 0, i32 0
  %50 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %48, i32 0, i32 1
  store i8* getelementptr inbounds ([15 x i8], [15 x i8]* @6, i32 0, i32 0), i8** %49
  store i32 14, i32* %50
  %51 = load { i8*, i32 }, { i8*, i32 }* %48
  %52 = extractvalue { i8*, i32 } %51, 0
  %53 = call i32 @puts(i8* %52)
  %54 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }*
%gaussian, i32 0, i32 0
  %55 = load double*, double** %54
  %56 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }*
%gaussian, i32 0, i32 1
  %57 = load i32, i32* %56
  %58 = mul i32 %57, 2
  %59 = add i32 %58, 2
  %60 = getelementptr double, double* %55, i32 %59
  %61 = load double, double* %60
  %62 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
@7, i32 0, i32 0), double %61)
  %63 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 },
{ double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img, i32 0,
i32 2
  %64 = load { double*, i32, i32 }, { double*, i32, i32 }* %63
  %gaussian2 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %65 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %64, { double*, i32, i32 }* %65
  %66 = extractvalue { double*, i32, i32 } %64, 1
  %67 = extractvalue { double*, i32, i32 } %64, 2
  %68 = mul i32 %66, %67
  %69 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian2, { double*, i32, i32 }* %69
  %70 = extractvalue { double*, i32, i32 } %gaussian2, 1
  %71 = extractvalue { double*, i32, i32 } %gaussian2, 2
  %72 = extractvalue { double*, i32, i32 } %64, 1
  %73 = extractvalue { double*, i32, i32 } %64, 2
  %74 = extractvalue { double*, i32, i32 } %gaussian2, 1
  %75 = mul i32 %72, %73
  %76 = alloca { double*, i32, i32 }
  %77 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %76, i32
0, i32 0
```

```
  %mallocsize = mul i32 %75, ptrtoint (double* getelementptr (double, double* null,
i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %78 = bitcast i8* %malloccall to double*
  %79 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %76, i32
0, i32 1
  %80 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %76, i32
0, i32 2
  store double* %78, double** %77
  store i32 %72, i32* %79
  store i32 %73, i32* %80
  call void @_mat_mat_convolute({ double*, i32, i32 }* %65, { double*, i32, i32 }*
%69, { double*, i32, i32 }* %76)
  %81 = load { double*, i32, i32 }, { double*, i32, i32 }* %76
  %82 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 },
{ double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img, i32 0,
i32 2
  store { double*, i32, i32 } %81, { double*, i32, i32 }* %82
  %83 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 },
{ double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img, i32 0,
i32 4
  %84 = load { double*, i32, i32 }, { double*, i32, i32 }* %83
  %gaussian3 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %85 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %84, { double*, i32, i32 }* %85
  %86 = extractvalue { double*, i32, i32 } %84, 1
  %87 = extractvalue { double*, i32, i32 } %84, 2
  %88 = mul i32 %86, %87
  %89 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian3, { double*, i32, i32 }* %89
  %90 = extractvalue { double*, i32, i32 } %gaussian3, 1
  %91 = extractvalue { double*, i32, i32 } %gaussian3, 2
  %92 = extractvalue { double*, i32, i32 } %84, 1
  %93 = extractvalue { double*, i32, i32 } %84, 2
  %94 = extractvalue { double*, i32, i32 } %gaussian3, 1
  %95 = mul i32 %92, %93
  %96 = alloca { double*, i32, i32 }
  %97 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %96, i32
0, i32 0
  %mallocsize4 = mul i32 %95, ptrtoint (double* getelementptr (double, double* null,
i32 1) to i32)
  %malloccall5 = tail call i8* @malloc(i32 %mallocsize4)
  %98 = bitcast i8* %malloccall5 to double*
  %99 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %96, i32
0, i32 1
  %100 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %96,
i32 0, i32 2
  store double* %98, double** %97
  store i32 %92, i32* %99
  store i32 %93, i32* %100
  call void @_mat_mat_convolute({ double*, i32, i32 }* %85, { double*, i32, i32 }*
%89, { double*, i32, i32 }* %96)
```

```
  %101 = load { double*, i32, i32 }, { double*, i32, i32 }* %96
  %102 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img,
i32 0, i32 4
  store { double*, i32, i32 } %101, { double*, i32, i32 }* %102
  %103 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img,
i32 0, i32 3
  %104 = load { double*, i32, i32 }, { double*, i32, i32 }* %103
  %gaussian6 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %105 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %104, { double*, i32, i32 }* %105
  %106 = extractvalue { double*, i32, i32 } %104, 1
  %107 = extractvalue { double*, i32, i32 } %104, 2
  %108 = mul i32 %106, %107
  %109 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian6, { double*, i32, i32 }* %109
  %110 = extractvalue { double*, i32, i32 } %gaussian6, 1
  %111 = extractvalue { double*, i32, i32 } %gaussian6, 2
  %112 = extractvalue { double*, i32, i32 } %104, 1
  %113 = extractvalue { double*, i32, i32 } %104, 2
  %114 = extractvalue { double*, i32, i32 } %gaussian6, 1
  %115 = mul i32 %112, %113
  %116 = alloca { double*, i32, i32 }
  %117 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %116,
i32 0, i32 0
  %mallocsize7 = mul i32 %115, ptrtoint (double* getelementptr (double, double* null,
i32 1) to i32)
  %malloccall8 = tail call i8* @malloc(i32 %mallocsize7)
  %118 = bitcast i8* %malloccall8 to double*
  %119 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %116,
i32 0, i32 1
  %120 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %116,
i32 0, i32 2
  store double* %118, double** %117
  store i32 %112, i32* %119
  store i32 %113, i32* %120
  call void @_mat_mat_convolute({ double*, i32, i32 }* %105, { double*, i32, i32 }*
%109, { double*, i32, i32 }* %116)
  %121 = load { double*, i32, i32 }, { double*, i32, i32 }* %116
  %122 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img,
i32 0, i32 3
  store { double*, i32, i32 } %121, { double*, i32, i32 }* %122
  %123 = alloca { i8*, i32 }
  %124 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %123, i32 0, i32 0
  %125 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %123, i32 0, i32 1
  store i8* getelementptr inbounds ([21 x i8], [21 x i8]* @8, i32 0, i32 0), i8**
%124
  store i32 20, i32* %125
  %126 = load { i8*, i32 }, { i8*, i32 }* %123
```

```
  %img9 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img
  %127 = extractvalue { i8*, i32 } %126, 0
  %128 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %img9, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %128
  call void @_image_write(i8* %127, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %128)
  %129 = alloca { i8*, i32 }
  %130 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %129, i32 0, i32 0
  %131 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %129, i32 0, i32 1
  store i8* getelementptr inbounds ([8 x i8], [8 x i8]* @9, i32 0, i32 0), i8** %130
  store i32 7, i32* %131
  %132 = load { i8*, i32 }, { i8*, i32 }* %129
  %133 = extractvalue { i8*, i32 } %132, 0
  %134 = call i32 @puts(i8* %133)
  %sharpen = alloca { double*, i32, i32 }
  %135 = alloca { double*, i32, i32 }
  %136 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %135,
i32 0, i32 0
  %137 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %135,
i32 0, i32 1
  %138 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %135,
i32 0, i32 2
  %139 = alloca double, i32 9
  store double* %139, double** %136
  store i32 3, i32* %137
  store i32 3, i32* %138
  call void @_mat_gen_sharpen({ double*, i32, i32 }* %135)
  %140 = load { double*, i32, i32 }, { double*, i32, i32 }* %135
  store { double*, i32, i32 } %140, { double*, i32, i32 }* %sharpen
  %sharpen10 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %141 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen10, { double*, i32, i32 }* %141
  call void @_mat_print({ double*, i32, i32 }* %141)
  %142 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2,
i32 0, i32 2
  %143 = load { double*, i32, i32 }, { double*, i32, i32 }* %142
  %sharpen11 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %144 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %143, { double*, i32, i32 }* %144
  %145 = extractvalue { double*, i32, i32 } %143, 1
  %146 = extractvalue { double*, i32, i32 } %143, 2
  %147 = mul i32 %145, %146
  %148 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen11, { double*, i32, i32 }* %148
  %149 = extractvalue { double*, i32, i32 } %sharpen11, 1
  %150 = extractvalue { double*, i32, i32 } %sharpen11, 2
  %151 = extractvalue { double*, i32, i32 } %143, 1
```

```
  %152 = extractvalue { double*, i32, i32 } %143, 2
  %153 = extractvalue { double*, i32, i32 } %sharpen11, 1
  %154 = mul i32 %151, %152
  %155 = alloca { double*, i32, i32 }
  %156 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %155,
i32 0, i32 0
  %mallocsize12 = mul i32 %154, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall13 = tail call i8* @malloc(i32 %mallocsize12)
  %157 = bitcast i8* %malloccall13 to double*
  %158 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %155,
i32 0, i32 1
  %159 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %155,
i32 0, i32 2
  store double* %157, double** %156
  store i32 %151, i32* %158
  store i32 %152, i32* %159
  call void @_mat_mat_convolute({ double*, i32, i32 }* %144, { double*, i32, i32 }*
%148, { double*, i32, i32 }* %155)
  %160 = load { double*, i32, i32 }, { double*, i32, i32 }* %155
  %161 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2,
i32 0, i32 2
  store { double*, i32, i32 } %160, { double*, i32, i32 }* %161
  %162 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2,
i32 0, i32 4
  %163 = load { double*, i32, i32 }, { double*, i32, i32 }* %162
  %sharpen14 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %164 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %163, { double*, i32, i32 }* %164
  %165 = extractvalue { double*, i32, i32 } %163, 1
  %166 = extractvalue { double*, i32, i32 } %163, 2
  %167 = mul i32 %165, %166
  %168 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen14, { double*, i32, i32 }* %168
  %169 = extractvalue { double*, i32, i32 } %sharpen14, 1
  %170 = extractvalue { double*, i32, i32 } %sharpen14, 2
  %171 = extractvalue { double*, i32, i32 } %163, 1
  %172 = extractvalue { double*, i32, i32 } %163, 2
  %173 = extractvalue { double*, i32, i32 } %sharpen14, 1
  %174 = mul i32 %171, %172
  %175 = alloca { double*, i32, i32 }
  %176 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %175,
i32 0, i32 0
  %mallocsize15 = mul i32 %174, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall16 = tail call i8* @malloc(i32 %mallocsize15)
  %177 = bitcast i8* %malloccall16 to double*
  %178 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %175,
i32 0, i32 1
```

```
  %179 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %175,
i32 0, i32 2
  store double* %177, double** %176
  store i32 %171, i32* %178
  store i32 %172, i32* %179
  call void @_mat_mat_convolute({ double*, i32, i32 }* %164, { double*, i32, i32 }*
%168, { double*, i32, i32 }* %175)
  %180 = load { double*, i32, i32 }, { double*, i32, i32 }* %175
  %181 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2,
i32 0, i32 4
  store { double*, i32, i32 } %180, { double*, i32, i32 }* %181
  %182 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2,
i32 0, i32 3
  %183 = load { double*, i32, i32 }, { double*, i32, i32 }* %182
  %sharpen17 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %184 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %183, { double*, i32, i32 }* %184
  %185 = extractvalue { double*, i32, i32 } %183, 1
  %186 = extractvalue { double*, i32, i32 } %183, 2
  %187 = mul i32 %185, %186
  %188 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen17, { double*, i32, i32 }* %188
  %189 = extractvalue { double*, i32, i32 } %sharpen17, 1
  %190 = extractvalue { double*, i32, i32 } %sharpen17, 2
  %191 = extractvalue { double*, i32, i32 } %183, 1
  %192 = extractvalue { double*, i32, i32 } %183, 2
  %193 = extractvalue { double*, i32, i32 } %sharpen17, 1
  %194 = mul i32 %191, %192
  %195 = alloca { double*, i32, i32 }
  %196 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %195,
i32 0, i32 0
  %mallocsize18 = mul i32 %194, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall19 = tail call i8* @malloc(i32 %mallocsize18)
  %197 = bitcast i8* %malloccall19 to double*
  %198 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %195,
i32 0, i32 1
  %199 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %195,
i32 0, i32 2
  store double* %197, double** %196
  store i32 %191, i32* %198
  store i32 %192, i32* %199
  call void @_mat_mat_convolute({ double*, i32, i32 }* %184, { double*, i32, i32 }*
%188, { double*, i32, i32 }* %195)
  %200 = load { double*, i32, i32 }, { double*, i32, i32 }* %195
  %201 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2,
i32 0, i32 3
  store { double*, i32, i32 } %200, { double*, i32, i32 }* %201
```

```
  %202 = alloca { i8*, i32 }
  %203 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %202, i32 0, i32 0
  %204 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %202, i32 0, i32 1
  store i8* getelementptr inbounds ([20 x i8], [20 x i8]* @10, i32 0, i32 0), i8**
%203
  store i32 19, i32* %204
  %205 = load { i8*, i32 }, { i8*, i32 }* %202
  %img220 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img2
  %206 = extractvalue { i8*, i32 } %205, 0
  %207 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %img220, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %207
  call void @_image_write(i8* %206, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %207)
  %208 = alloca { i8*, i32 }
  %209 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %208, i32 0, i32 0
  %210 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %208, i32 0, i32 1
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @11, i32 0, i32 0), i8**
%209
  store i32 11, i32* %210
  %211 = load { i8*, i32 }, { i8*, i32 }* %208
  %212 = extractvalue { i8*, i32 } %211, 0
  %213 = call i32 @puts(i8* %212)
  %edge = alloca { double*, i32, i32 }
  %214 = alloca { double*, i32, i32 }
  %215 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %214,
i32 0, i32 0
  %216 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %214,
i32 0, i32 1
  %217 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %214,
i32 0, i32 2
  %218 = alloca double, i32 9
  store double* %218, double** %215
  store i32 3, i32* %216
  store i32 3, i32* %217
  call void @_mat_gen_edge_detect({ double*, i32, i32 }* %214)
 %219 = load { double*, i32, i32 }, { double*, i32, i32 }* %214
  store { double*, i32, i32 } %219, { double*, i32, i32 }* %edge
  %edge21 = load { double*, i32, i32 }, { double*, i32, i32 }* %edge
  %220 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %edge21, { double*, i32, i32 }* %220
  call void @_mat_print({ double*, i32, i32 }* %220)
  %221 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3,
i32 0, i32 2
  %222 = load { double*, i32, i32 }, { double*, i32, i32 }* %221
  %sharpen22 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %223 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %222, { double*, i32, i32 }* %223
```

```
  %224 = extractvalue { double*, i32, i32 } %222, 1
  %225 = extractvalue { double*, i32, i32 } %222, 2
  %226 = mul i32 %224, %225
  %227 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen22, { double*, i32, i32 }* %227
  %228 = extractvalue { double*, i32, i32 } %sharpen22, 1
  %229 = extractvalue { double*, i32, i32 } %sharpen22, 2
  %230 = extractvalue { double*, i32, i32 } %222, 1
  %231 = extractvalue { double*, i32, i32 } %222, 2
  %232 = extractvalue { double*, i32, i32 } %sharpen22, 1
  %233 = mul i32 %230, %231
  %234 = alloca { double*, i32, i32 }
  %235 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %234,
i32 0, i32 0
  %mallocsize23 = mul i32 %233, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall24 = tail call i8* @malloc(i32 %mallocsize23)
  %236 = bitcast i8* %malloccall24 to double*
  %237 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %234,
i32 0, i32 1
  %238 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %234,
i32 0, i32 2
  store double* %236, double** %235
  store i32 %230, i32* %237
  store i32 %231, i32* %238
  call void @_mat_mat_convolute({ double*, i32, i32 }* %223, { double*, i32, i32 }*
%227, { double*, i32, i32 }* %234)
  %239 = load { double*, i32, i32 }, { double*, i32, i32 }* %234
  %edge25 = load { double*, i32, i32 }, { double*, i32, i32 }* %edge
  %240 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %239, { double*, i32, i32 }* %240
  %241 = extractvalue { double*, i32, i32 } %239, 1
  %242 = extractvalue { double*, i32, i32 } %239, 2
  %243 = mul i32 %241, %242
  %244 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %edge25, { double*, i32, i32 }* %244
  %245 = extractvalue { double*, i32, i32 } %edge25, 1
  %246 = extractvalue { double*, i32, i32 } %edge25, 2
  %247 = extractvalue { double*, i32, i32 } %239, 1
  %248 = extractvalue { double*, i32, i32 } %239, 2
  %249 = extractvalue { double*, i32, i32 } %edge25, 1
  %250 = mul i32 %247, %248
  %251 = alloca { double*, i32, i32 }
  %252 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %251,
i32 0, i32 0
  %mallocsize26 = mul i32 %250, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall27 = tail call i8* @malloc(i32 %mallocsize26)
  %253 = bitcast i8* %malloccall27 to double*
  %254 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %251,
i32 0, i32 1
  %255 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %251,
i32 0, i32 2
  store double* %253, double** %252
```

```
  store i32 %247, i32* %254
  store i32 %248, i32* %255
  call void @_mat_mat_convolute({ double*, i32, i32 }* %240, { double*, i32, i32 }*
%244, { double*, i32, i32 }* %251)
  %256 = load { double*, i32, i32 }, { double*, i32, i32 }* %251
  %257 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3,
i32 0, i32 2
  store { double*, i32, i32 } %256, { double*, i32, i32 }* %257
  %258 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3,
i32 0, i32 4
  %259 = load { double*, i32, i32 }, { double*, i32, i32 }* %258
  %sharpen28 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %260 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %259, { double*, i32, i32 }* %260
  %261 = extractvalue { double*, i32, i32 } %259, 1
  %262 = extractvalue { double*, i32, i32 } %259, 2
  %263 = mul i32 %261, %262
  %264 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen28, { double*, i32, i32 }* %264
  %265 = extractvalue { double*, i32, i32 } %sharpen28, 1
  %266 = extractvalue { double*, i32, i32 } %sharpen28, 2
  %267 = extractvalue { double*, i32, i32 } %259, 1
  %268 = extractvalue { double*, i32, i32 } %259, 2
  %269 = extractvalue { double*, i32, i32 } %sharpen28, 1
  %270 = mul i32 %267, %268
  %271 = alloca { double*, i32, i32 }
  %272 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %271,
i32 0, i32 0
  %mallocsize29 = mul i32 %270, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall30 = tail call i8* @malloc(i32 %mallocsize29)
  %273 = bitcast i8* %malloccall30 to double*
  %274 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %271,
i32 0, i32 1
  %275 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %271,
i32 0, i32 2
  store double* %273, double** %272
  store i32 %267, i32* %274
  store i32 %268, i32* %275
  call void @_mat_mat_convolute({ double*, i32, i32 }* %260, { double*, i32, i32 }*
%264, { double*, i32, i32 }* %271)
  %276 = load { double*, i32, i32 }, { double*, i32, i32 }* %271
  %edge31 = load { double*, i32, i32 }, { double*, i32, i32 }* %edge
  %277 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %276, { double*, i32, i32 }* %277
  %278 = extractvalue { double*, i32, i32 } %276, 1
  %279 = extractvalue { double*, i32, i32 } %276, 2
  %280 = mul i32 %278, %279
  %281 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %edge31, { double*, i32, i32 }* %281
```

```
  %282 = extractvalue { double*, i32, i32 } %edge31, 1
  %283 = extractvalue { double*, i32, i32 } %edge31, 2
  %284 = extractvalue { double*, i32, i32 } %276, 1
  %285 = extractvalue { double*, i32, i32 } %276, 2
  %286 = extractvalue { double*, i32, i32 } %edge31, 1
  %287 = mul i32 %284, %285
  %288 = alloca { double*, i32, i32 }
  %289 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %288,
i32 0, i32 0
  %mallocsize32 = mul i32 %287, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall33 = tail call i8* @malloc(i32 %mallocsize32)
  %290 = bitcast i8* %malloccall33 to double*
  %291 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %288,
i32 0, i32 1
  %292 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %288,
i32 0, i32 2
  store double* %290, double** %289
  store i32 %284, i32* %291
  store i32 %285, i32* %292
  call void @_mat_mat_convolute({ double*, i32, i32 }* %277, { double*, i32, i32 }*
%281, { double*, i32, i32 }* %288)
  %293 = load { double*, i32, i32 }, { double*, i32, i32 }* %288
  %294 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3,
i32 0, i32 4
  store { double*, i32, i32 } %293, { double*, i32, i32 }* %294
  %295 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3,
i32 0, i32 3
  %296 = load { double*, i32, i32 }, { double*, i32, i32 }* %295
  %sharpen34 = load { double*, i32, i32 }, { double*, i32, i32 }* %sharpen
  %297 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %296, { double*, i32, i32 }* %297
  %298 = extractvalue { double*, i32, i32 } %296, 1
  %299 = extractvalue { double*, i32, i32 } %296, 2
  %300 = mul i32 %298, %299
  %301 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %sharpen34, { double*, i32, i32 }* %301
  %302 = extractvalue { double*, i32, i32 } %sharpen34, 1
  %303 = extractvalue { double*, i32, i32 } %sharpen34, 2
  %304 = extractvalue { double*, i32, i32 } %296, 1
  %305 = extractvalue { double*, i32, i32 } %296, 2
  %306 = extractvalue { double*, i32, i32 } %sharpen34, 1
  %307 = mul i32 %304, %305
  %308 = alloca { double*, i32, i32 }
  %309 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %308,
i32 0, i32 0
  %mallocsize35 = mul i32 %307, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall36 = tail call i8* @malloc(i32 %mallocsize35)
  %310 = bitcast i8* %malloccall36 to double*
```

```
  %311 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %308,
i32 0, i32 1
  %312 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %308,
i32 0, i32 2
  store double* %310, double** %309
  store i32 %304, i32* %311
  store i32 %305, i32* %312
  call void @_mat_mat_convolute({ double*, i32, i32 }* %297, { double*, i32, i32 }*
%301, { double*, i32, i32 }* %308)
  %313 = load { double*, i32, i32 }, { double*, i32, i32 }* %308
  %edge37 = load { double*, i32, i32 }, { double*, i32, i32 }* %edge
  %314 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %313, { double*, i32, i32 }* %314
  %315 = extractvalue { double*, i32, i32 } %313, 1
  %316 = extractvalue { double*, i32, i32 } %313, 2
  %317 = mul i32 %315, %316
  %318 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %edge37, { double*, i32, i32 }* %318
  %319 = extractvalue { double*, i32, i32 } %edge37, 1
  %320 = extractvalue { double*, i32, i32 } %edge37, 2
  %321 = extractvalue { double*, i32, i32 } %313, 1
  %322 = extractvalue { double*, i32, i32 } %313, 2
  %323 = extractvalue { double*, i32, i32 } %edge37, 1
  %324 = mul i32 %321, %322
  %325 = alloca { double*, i32, i32 }
  %326 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %325,
i32 0, i32 0
  %mallocsize38 = mul i32 %324, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall39 = tail call i8* @malloc(i32 %mallocsize38)
  %327 = bitcast i8* %malloccall39 to double*
  %328 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %325,
i32 0, i32 1
  %329 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %325,
i32 0, i32 2
  store double* %327, double** %326
  store i32 %321, i32* %328
  store i32 %322, i32* %329
  call void @_mat_mat_convolute({ double*, i32, i32 }* %314, { double*, i32, i32 }*
%318, { double*, i32, i32 }* %325)
  %330 = load { double*, i32, i32 }, { double*, i32, i32 }* %325
  %331 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3,
i32 0, i32 3
  store { double*, i32, i32 } %330, { double*, i32, i32 }* %331
  %332 = alloca { i8*, i32 }
  %333 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %332, i32 0, i32 0
  %334 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %332, i32 0, i32 1
  store i8* getelementptr inbounds ([24 x i8], [24 x i8]* @12, i32 0, i32 0), i8**
%333
  store i32 23, i32* %334
  %335 = load { i8*, i32 }, { i8*, i32 }* %332
```

```
  %img340 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3
  %336 = extractvalue { i8*, i32 } %335, 0
  %337 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %img340, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %337
  call void @_image_write(i8* %336, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %337)
  %brighten = alloca { double*, i32, i32 }
  %338 = alloca { double*, i32, i32 }
  %339 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %338,
i32 0, i32 0
  %340 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %338,
i32 0, i32 1
  %341 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %338,
i32 0, i32 2
  %342 = alloca double, i32 9
  store double* %342, double** %339
  store i32 3, i32* %340
  store i32 3, i32* %341
  call void @_mat_gen_brighten(double 1.500000e+00, { double*, i32, i32 }* %338)
  %343 = load { double*, i32, i32 }, { double*, i32, i32 }* %338
  store { double*, i32, i32 } %343, { double*, i32, i32 }* %brighten
  %brighten41 = load { double*, i32, i32 }, { double*, i32, i32 }* %brighten
  %344 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %brighten41, { double*, i32, i32 }* %344
  call void @_mat_print({ double*, i32, i32 }* %344)
  %345 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4,
i32 0, i32 2
  %346 = load { double*, i32, i32 }, { double*, i32, i32 }* %345
  %brighten42 = load { double*, i32, i32 }, { double*, i32, i32 }* %brighten
  %347 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %346, { double*, i32, i32 }* %347
  %348 = extractvalue { double*, i32, i32 } %346, 1
  %349 = extractvalue { double*, i32, i32 } %346, 2
  %350 = mul i32 %348, %349
  %351 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %brighten42, { double*, i32, i32 }* %351
  %352 = extractvalue { double*, i32, i32 } %brighten42, 1
  %353 = extractvalue { double*, i32, i32 } %brighten42, 2
  %354 = extractvalue { double*, i32, i32 } %346, 1
  %355 = extractvalue { double*, i32, i32 } %346, 2
  %356 = extractvalue { double*, i32, i32 } %brighten42, 1
  %357 = mul i32 %354, %355
  %358 = alloca { double*, i32, i32 }
  %359 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %358,
i32 0, i32 0
  %mallocsize43 = mul i32 %357, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
```

```
  %malloccall44 = tail call i8* @malloc(i32 %mallocsize43)
  %360 = bitcast i8* %malloccall44 to double*
  %361 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %358,
i32 0, i32 1
  %362 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %358,
i32 0, i32 2
  store double* %360, double** %359
  store i32 %354, i32* %361
  store i32 %355, i32* %362
  call void @_mat_mat_convolute({ double*, i32, i32 }* %347, { double*, i32, i32 }*
%351, { double*, i32, i32 }* %358)
  %363 = load { double*, i32, i32 }, { double*, i32, i32 }* %358
  %gaussian45 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %364 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %363, { double*, i32, i32 }* %364
  %365 = extractvalue { double*, i32, i32 } %363, 1
  %366 = extractvalue { double*, i32, i32 } %363, 2
  %367 = mul i32 %365, %366
  %368 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian45, { double*, i32, i32 }* %368
  %369 = extractvalue { double*, i32, i32 } %gaussian45, 1
  %370 = extractvalue { double*, i32, i32 } %gaussian45, 2
  %371 = extractvalue { double*, i32, i32 } %363, 1
  %372 = extractvalue { double*, i32, i32 } %363, 2
  %373 = extractvalue { double*, i32, i32 } %gaussian45, 1
  %374 = mul i32 %371, %372
  %375 = alloca { double*, i32, i32 }
  %376 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %375,
i32 0, i32 0
  %mallocsize46 = mul i32 %374, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall47 = tail call i8* @malloc(i32 %mallocsize46)
  %377 = bitcast i8* %malloccall47 to double*
  %378 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %375,
i32 0, i32 1
  %379 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %375,
i32 0, i32 2
  store double* %377, double** %376
  store i32 %371, i32* %378
  store i32 %372, i32* %379
  call void @_mat_mat_convolute({ double*, i32, i32 }* %364, { double*, i32, i32 }*
%368, { double*, i32, i32 }* %375)
  %380 = load { double*, i32, i32 }, { double*, i32, i32 }* %375
  %381 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4,
i32 0, i32 2
  store { double*, i32, i32 } %380, { double*, i32, i32 }* %381
  %382 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4,
i32 0, i32 4
  %383 = load { double*, i32, i32 }, { double*, i32, i32 }* %382
  %brighten48 = load { double*, i32, i32 }, { double*, i32, i32 }* %brighten
```

```
  %384 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %383, { double*, i32, i32 }* %384
  %385 = extractvalue { double*, i32, i32 } %383, 1
  %386 = extractvalue { double*, i32, i32 } %383, 2
  %387 = mul i32 %385, %386
  %388 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %brighten48, { double*, i32, i32 }* %388
  %389 = extractvalue { double*, i32, i32 } %brighten48, 1
  %390 = extractvalue { double*, i32, i32 } %brighten48, 2
  %391 = extractvalue { double*, i32, i32 } %383, 1
  %392 = extractvalue { double*, i32, i32 } %383, 2
  %393 = extractvalue { double*, i32, i32 } %brighten48, 1
  %394 = mul i32 %391, %392
  %395 = alloca { double*, i32, i32 }
  %396 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %395,
i32 0, i32 0
  %mallocsize49 = mul i32 %394, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall50 = tail call i8* @malloc(i32 %mallocsize49)
  %397 = bitcast i8* %malloccall50 to double*
  %398 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %395,
i32 0, i32 1
  %399 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %395,
i32 0, i32 2
  store double* %397, double** %396
  store i32 %391, i32* %398
  store i32 %392, i32* %399
  call void @_mat_mat_convolute({ double*, i32, i32 }* %384, { double*, i32, i32 }*
%388, { double*, i32, i32 }* %395)
  %400 = load { double*, i32, i32 }, { double*, i32, i32 }* %395
  %gaussian51 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %401 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %400, { double*, i32, i32 }* %401
  %402 = extractvalue { double*, i32, i32 } %400, 1
  %403 = extractvalue { double*, i32, i32 } %400, 2
  %404 = mul i32 %402, %403
  %405 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian51, { double*, i32, i32 }* %405
  %406 = extractvalue { double*, i32, i32 } %gaussian51, 1
  %407 = extractvalue { double*, i32, i32 } %gaussian51, 2
  %408 = extractvalue { double*, i32, i32 } %400, 1
  %409 = extractvalue { double*, i32, i32 } %400, 2
  %410 = extractvalue { double*, i32, i32 } %gaussian51, 1
  %411 = mul i32 %408, %409
  %412 = alloca { double*, i32, i32 }
  %413 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %412,
i32 0, i32 0
  %mallocsize52 = mul i32 %411, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall53 = tail call i8* @malloc(i32 %mallocsize52)
  %414 = bitcast i8* %malloccall53 to double*
  %415 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %412,
i32 0, i32 1
```

```
  %416 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %412,
i32 0, i32 2
  store double* %414, double** %413
  store i32 %408, i32* %415
  store i32 %409, i32* %416
  call void @_mat_mat_convolute({ double*, i32, i32 }* %401, { double*, i32, i32 }*
%405, { double*, i32, i32 }* %412)
  %417 = load { double*, i32, i32 }, { double*, i32, i32 }* %412
  %418 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4,
i32 0, i32 4
  store { double*, i32, i32 } %417, { double*, i32, i32 }* %418
  %419 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4,
i32 0, i32 3
  %420 = load { double*, i32, i32 }, { double*, i32, i32 }* %419
  %brighten54 = load { double*, i32, i32 }, { double*, i32, i32 }* %brighten
  %421 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %420, { double*, i32, i32 }* %421
  %422 = extractvalue { double*, i32, i32 } %420, 1
  %423 = extractvalue { double*, i32, i32 } %420, 2
  %424 = mul i32 %422, %423
  %425 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %brighten54, { double*, i32, i32 }* %425
  %426 = extractvalue { double*, i32, i32 } %brighten54, 1
  %427 = extractvalue { double*, i32, i32 } %brighten54, 2
  %428 = extractvalue { double*, i32, i32 } %420, 1
  %429 = extractvalue { double*, i32, i32 } %420, 2
  %430 = extractvalue { double*, i32, i32 } %brighten54, 1
  %431 = mul i32 %428, %429
  %432 = alloca { double*, i32, i32 }
  %433 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %432,
i32 0, i32 0
  %mallocsize55 = mul i32 %431, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall56 = tail call i8* @malloc(i32 %mallocsize55)
  %434 = bitcast i8* %malloccall56 to double*
  %435 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %432,
i32 0, i32 1
  %436 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %432,
i32 0, i32 2
  store double* %434, double** %433
  store i32 %428, i32* %435
  store i32 %429, i32* %436
  call void @_mat_mat_convolute({ double*, i32, i32 }* %421, { double*, i32, i32 }*
%425, { double*, i32, i32 }* %432)
  %437 = load { double*, i32, i32 }, { double*, i32, i32 }* %432
  %gaussian57 = load { double*, i32, i32 }, { double*, i32, i32 }* %gaussian
  %438 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %437, { double*, i32, i32 }* %438
  %439 = extractvalue { double*, i32, i32 } %437, 1
  %440 = extractvalue { double*, i32, i32 } %437, 2
```

```
  %441 = mul i32 %439, %440
  %442 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %gaussian57, { double*, i32, i32 }* %442
  %443 = extractvalue { double*, i32, i32 } %gaussian57, 1
  %444 = extractvalue { double*, i32, i32 } %gaussian57, 2
  %445 = extractvalue { double*, i32, i32 } %437, 1
  %446 = extractvalue { double*, i32, i32 } %437, 2
  %447 = extractvalue { double*, i32, i32 } %gaussian57, 1
  %448 = mul i32 %445, %446
  %449 = alloca { double*, i32, i32 }
  %450 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %449,
i32 0, i32 0
  %mallocsize58 = mul i32 %448, ptrtoint (double* getelementptr (double, double*
null, i32 1) to i32)
  %malloccall59 = tail call i8* @malloc(i32 %mallocsize58)
  %451 = bitcast i8* %malloccall59 to double*
  %452 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %449,
i32 0, i32 1
  %453 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %449,
i32 0, i32 2
  store double* %451, double** %450
  store i32 %445, i32* %452
  store i32 %446, i32* %453
  call void @_mat_mat_convolute({ double*, i32, i32 }* %438, { double*, i32, i32 }*
%442, { double*, i32, i32 }* %449)
  %454 = load { double*, i32, i32 }, { double*, i32, i32 }* %449
  %455 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img4,
i32 0, i32 3
  store { double*, i32, i32 } %454, { double*, i32, i32 }* %455
  %456 = alloca { i8*, i32 }
  %457 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %456, i32 0, i32 0
  %458 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %456, i32 0, i32 1
  store i8* getelementptr inbounds ([28 x i8], [28 x i8]* @13, i32 0, i32 0), i8**
%457
  store i32 27, i32* %458
  %459 = load { i8*, i32 }, { i8*, i32 }* %456
  %img360 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img3
  %460 = extractvalue { i8*, i32 } %459, 0
  %461 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %img360, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %461
  call void @_image_write(i8* %460, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %461)
  %462 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img5,
i32 0, i32 2
  %463 = load { double*, i32, i32 }, { double*, i32, i32 }* %462
```

```
  %464 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img5,
i32 0, i32 3
  store { double*, i32, i32 } %463, { double*, i32, i32 }* %464
  %465 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img5,
i32 0, i32 4
  store { double*, i32, i32 } %463, { double*, i32, i32 }* %465
  %466 = alloca { i8*, i32 }
  %467 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %466, i32 0, i32 0
  %468 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %466, i32 0, i32 1
  store i8* getelementptr inbounds ([22 x i8], [22 x i8]* @14, i32 0, i32 0), i8**
%467
  store i32 21, i32* %468
  %469 = load { i8*, i32 }, { i8*, i32 }* %466
  %img561 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img5
  %470 = extractvalue { i8*, i32 } %469, 0
  %471 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %img561, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %471
  call void @_image_write(i8* %470, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %471)
  ret i32 0
}

declare noalias i8* @malloc(i32)
```

6.2.3 Program 2

```
string name = "square";
string ext = "png";
string file = name + "." + ext;
Image img = coload(file);
int h = height(img@red);
matrix result = new(h, h);
int i; int j;
for i = 0; i < h; i = i + 1; {
      for j = 0; j < h; j = j + 1;  {
            float sum = 0.0;
            int k;
            for k = 0; k < h; k = k + 1; {
                  float addendum = img@red.i|k * img@blue.k|j;
                  sum = sum+addendum;
```

```
            }
            result.i|j = sum;
      }
}
mprint(result);
print("Output of standard library matrix multiplication:");
mprint(img@blue * img@red);
```

6.3.4 LLVM Output for Program 2

```
; ModuleID = 'Colode'
source_filename = "Colode"

@0 = private unnamed_addr constant [7 x i8] c"square\00"
@1 = private unnamed_addr constant [4 x i8] c"png\00"
@2 = private unnamed_addr constant [2 x i8] c".\00"
@3 = private unnamed_addr constant [50 x i8] c"Output of standard library matrix
multiplication:\00"

declare i32 @puts(i8*)

declare double @pow(double, double)

declare i32 @printf(i8*, ...)

declare void @_mat_zero_out({ double*, i32, i32 }*)

declare void @_mat_print({ double*, i32, i32 }*)

declare void @_mat_scalar_add({ double*, i32, i32 }*, double, { double*, i32, i32 }*)

declare void @_mat_scalar_subtract({ double*, i32, i32 }*, double, { double*, i32,
i32 }*)

declare void @_mat_scalar_multiply({ double*, i32, i32 }*, double, { double*, i32,
i32 }*)

declare void @_mat_scalar_divide({ double*, i32, i32 }*, double, { double*, i32, i32
}*)

declare void @_mat_mat_add({ double*, i32, i32 }*, { double*, i32, i32 }*, { double*,
i32, i32 }*)

declare void @_mat_mat_subtract({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare void @_mat_mat_multiply({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)
```

```
declare void @_mat_mat_divide({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare void @_mat_mat_convolute({ double*, i32, i32 }*, { double*, i32, i32 }*, {
double*, i32, i32 }*)

declare i1 @_mat_mat_equal({ double*, i32, i32 }*, { double*, i32, i32 }*)

declare void @_mat_mat_power({ double*, i32, i32 }*, i32, { double*, i32, i32 }*)

declare void @_image_read(i8*, { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }*)

declare void @_image_write(i8*, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }*)

declare void @_mat_gen_gauss(double, { double*, i32, i32 }*)

declare void @_mat_gen_sharpen({ double*, i32, i32 }*)

declare void @_mat_gen_edge_detect({ double*, i32, i32 }*)

declare void @_mat_gen_brighten(double, { double*, i32, i32 }*)

define i32 @main() {
entry:
  %name = alloca { i8*, i32 }
  %0 = alloca { i8*, i32 }
  %1 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 0
  %2 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 1
  store i8* getelementptr inbounds ([7 x i8], [7 x i8]* @0, i32 0, i32 0), i8** %1
  store i32 6, i32* %2
  %3 = load { i8*, i32 }, { i8*, i32 }* %0
  store { i8*, i32 } %3, { i8*, i32 }* %name
  %ext = alloca { i8*, i32 }
  %4 = alloca { i8*, i32 }
  %5 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %4, i32 0, i32 0
  %6 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %4, i32 0, i32 1
  store i8* getelementptr inbounds ([4 x i8], [4 x i8]* @1, i32 0, i32 0), i8** %5
  store i32 3, i32* %6
  %7 = load { i8*, i32 }, { i8*, i32 }* %4
  store { i8*, i32 } %7, { i8*, i32 }* %ext
  %file = alloca { i8*, i32 }
  %name1 = load { i8*, i32 }, { i8*, i32 }* %name
  %8 = alloca { i8*, i32 }
  %9 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %8, i32 0, i32 0
  %10 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %8, i32 0, i32 1
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @2, i32 0, i32 0), i8** %9
  store i32 1, i32* %10
  %11 = load { i8*, i32 }, { i8*, i32 }* %8
  %12 = alloca { i8*, i32 }
  %13 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %12, i32 0, i32 0
  %14 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %12, i32 0, i32 1
  %15 = extractvalue { i8*, i32 } %name1, 0
```

```
  %16 = extractvalue { i8*, i32 } %name1, 1
  %17 = extractvalue { i8*, i32 } %11, 0
  %18 = extractvalue { i8*, i32 } %11, 1
  %19 = add i32 %16, %18
  %20 = icmp eq i32 %19, 0
  %21 = alloca i8, i32 %19
  br i1 %20, label %if_then, label %if_else

if_merge:                                         ; preds = %if_else, %if_then
  %22 = load i8*, i8** %13
  %iter = alloca i32
  store i32 0, i32* %iter
  br label %while

if_then:                                          ; preds = %entry
  store i8* null, i8** %13
  br label %if_merge

if_else:                                          ; preds = %entry
  store i8* %21, i8** %13
  br label %if_merge

while:                                            ; preds = %while_body, %if_merge
  %23 = load i32, i32* %iter
  %24 = icmp slt i32 %23, %19
  br i1 %24, label %while_body, label %while_merge

while_body:                                       ; preds = %while
  %25 = load i32, i32* %iter
  %26 = icmp slt i32 %25, %16
  %27 = getelementptr i8, i8* %15, i32 %25
  %28 = sub i32 %25, %16
  %29 = getelementptr i8, i8* %17, i32 %28
  %30 = select i1 %26, i8* %27, i8* %29
  %31 = load i8, i8* %30
  %32 = getelementptr i8, i8* %22, i32 %25
  store i8 %31, i8* %32
  %33 = add i32 %25, 1
  store i32 %33, i32* %iter
  br label %while

while_merge:                                      ; preds = %while
  store i8* %22, i8** %13
  store i32 %19, i32* %14
  %34 = load { i8*, i32 }, { i8*, i32 }* %12
  %ext2 = load { i8*, i32 }, { i8*, i32 }* %ext
  %35 = alloca { i8*, i32 }
  %36 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %35, i32 0, i32 0
  %37 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %35, i32 0, i32 1
  %38 = extractvalue { i8*, i32 } %34, 0
  %39 = extractvalue { i8*, i32 } %34, 1
  %40 = extractvalue { i8*, i32 } %ext2, 0
  %41 = extractvalue { i8*, i32 } %ext2, 1
  %42 = add i32 %39, %41
```

```
  %43 = icmp eq i32 %42, 0
  %44 = alloca i8, i32 %42
  br i1 %43, label %if_then4, label %if_else5

if_merge3:                                        ; preds = %if_else5, %if_then4
  %45 = load i8*, i8** %36
  %iter6 = alloca i32
  store i32 0, i32* %iter6
  br label %while7

if_then4:                                         ; preds = %while_merge
  store i8* null, i8** %36
  br label %if_merge3

if_else5:                                         ; preds = %while_merge
  store i8* %44, i8** %36
  br label %if_merge3

while7:                                           ; preds = %while_body8, %if_merge3
  %46 = load i32, i32* %iter6
  %47 = icmp slt i32 %46, %42
  br i1 %47, label %while_body8, label %while_merge9

while_body8:                                      ; preds = %while7
  %48 = load i32, i32* %iter6
  %49 = icmp slt i32 %48, %39
  %50 = getelementptr i8, i8* %38, i32 %48
  %51 = sub i32 %48, %39
  %52 = getelementptr i8, i8* %40, i32 %51
  %53 = select i1 %49, i8* %50, i8* %52
  %54 = load i8, i8* %53
  %55 = getelementptr i8, i8* %45, i32 %48
  store i8 %54, i8* %55
  %56 = add i32 %48, 1
  store i32 %56, i32* %iter6
  br label %while7

while_merge9:                                     ; preds = %while7
  store i8* %45, i8** %36
  store i32 %42, i32* %37
  %57 = load { i8*, i32 }, { i8*, i32 }* %35
  store { i8*, i32 } %57, { i8*, i32 }* %file
  %img = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  %file10 = load { i8*, i32 }, { i8*, i32 }* %file
  %58 = extractvalue { i8*, i32 } %file10, 0
  %59 = alloca { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }
  call void @_image_read(i8* %58, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %59)
  %60 = load { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 }, { double*,
i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %59
```

```
  store { i32, i32, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 } } %60, { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img
  %h = alloca i32
  %61 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 },
{ double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img, i32 0,
i32 2
  %62 = load { double*, i32, i32 }, { double*, i32, i32 }* %61
  %63 = extractvalue { double*, i32, i32 } %62, 2
  store i32 %63, i32* %h
  %result = alloca { double*, i32, i32 }
  %h11 = load i32, i32* %h
  %h12 = load i32, i32* %h
  %64 = mul i32 %h11, %h12
  %65 = alloca { double*, i32, i32 }
  %66 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %65, i32
0, i32 0
  %67 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %65, i32
0, i32 1
  %68 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %65, i32
0, i32 2
  %69 = alloca double, i32 %64
  store double* %69, double** %66
  store i32 %h11, i32* %67
  store i32 %h12, i32* %68
  call void @_mat_zero_out({ double*, i32, i32 }* %65)
  %70 = load { double*, i32, i32 }, { double*, i32, i32 }* %65
  store { double*, i32, i32 } %70, { double*, i32, i32 }* %result
  %red = alloca { double*, i32, i32 }
  %71 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 },
{ double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img, i32 0,
i32 2
  %72 = load { double*, i32, i32 }, { double*, i32, i32 }* %71
  store { double*, i32, i32 } %72, { double*, i32, i32 }* %red
  %blue = alloca { double*, i32, i32 }
  %73 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32, i32
}, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32, i32 },
{ double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img, i32 0,
i32 4
  %74 = load { double*, i32, i32 }, { double*, i32, i32 }* %73
  store { double*, i32, i32 } %74, { double*, i32, i32 }* %blue
  %i = alloca i32
  %j = alloca i32
  store i32 0, i32* %i
  br label %while13

while13:                                          ; preds = %merge34, %while_merge9
  %i36 = load i32, i32* %i
  %h37 = load i32, i32* %h
  %75 = icmp slt i32 %i36, %h37
  br i1 %75, label %while_body14, label %merge38
```

```
while_body14:                                     ; preds = %while13
  store i32 0, i32* %j
  br label %while15

while15:                                          ; preds = %merge, %while_body14
  %j32 = load i32, i32* %j
  %h33 = load i32, i32* %h
  %76 = icmp slt i32 %j32, %h33
  br i1 %76, label %while_body16, label %merge34

while_body16:                                     ; preds = %while15
  %sum = alloca double
  store double 0.000000e+00, double* %sum
  %k = alloca i32
  store i32 0, i32* %k
  br label %while17

while17:                                          ; preds = %while_body18,
%while_body16
  %k26 = load i32, i32* %k
  %h27 = load i32, i32* %h
  %77 = icmp slt i32 %k26, %h27
  br i1 %77, label %while_body18, label %merge

while_body18:                                     ; preds = %while17
  %addendum = alloca double
  %78 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %red,
i32 0, i32 0
  %79 = load double*, double** %78
  %i19 = load i32, i32* %i
  %k20 = load i32, i32* %k
  %80 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %red,
i32 0, i32 1
  %81 = load i32, i32* %80
  %82 = mul i32 %81, %i19
  %83 = add i32 %82, %k20
  %84 = getelementptr double, double* %79, i32 %83
  %85 = load double, double* %84
  %86 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %blue,
i32 0, i32 0
  %87 = load double*, double** %86
  %k21 = load i32, i32* %k
  %j22 = load i32, i32* %j
  %88 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %blue,
i32 0, i32 1
  %89 = load i32, i32* %88
  %90 = mul i32 %89, %k21
  %91 = add i32 %90, %j22
  %92 = getelementptr double, double* %87, i32 %91
  %93 = load double, double* %92
  %94 = fmul double %85, %93
  store double %94, double* %addendum
  %sum23 = load double, double* %sum
  %addendum24 = load double, double* %addendum
```

```
  %95 = fadd double %sum23, %addendum24
  store double %95, double* %sum
  %k25 = load i32, i32* %k
  %96 = add i32 %k25, 1
  store i32 %96, i32* %k
  br label %while17

merge:                                            ; preds = %while17
  %sum28 = load double, double* %sum
  %97 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %result,
i32 0, i32 0
  %98 = load double*, double** %97
  %i29 = load i32, i32* %i
  %j30 = load i32, i32* %j
  %99 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %result,
i32 0, i32 1
  %100 = load i32, i32* %99
  %101 = mul i32 %100, %i29
  %102 = add i32 %101, %j30
  %103 = getelementptr double, double* %98, i32 %102
  store double %sum28, double* %103
  %j31 = load i32, i32* %j
  %104 = add i32 %j31, 1
  store i32 %104, i32* %j
  br label %while15

merge34:                                          ; preds = %while15
  %i35 = load i32, i32* %i
  %105 = add i32 %i35, 1
  store i32 %105, i32* %i
  br label %while13

merge38:                                          ; preds = %while13
  %result39 = load { double*, i32, i32 }, { double*, i32, i32 }* %result
  %106 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %result39, { double*, i32, i32 }* %106
  call void @_mat_print({ double*, i32, i32 }* %106)
  %107 = alloca { i8*, i32 }
  %108 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %107, i32 0, i32 0
  %109 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %107, i32 0, i32 1
  store i8* getelementptr inbounds ([50 x i8], [50 x i8]* @3, i32 0, i32 0), i8**
%108
  store i32 49, i32* %109
  %110 = load { i8*, i32 }, { i8*, i32 }* %107
  %111 = extractvalue { i8*, i32 } %110, 0
  %112 = call i32 @puts(i8* %111)
  %113 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img,
i32 0, i32 4
  %114 = load { double*, i32, i32 }, { double*, i32, i32 }* %113
  %115 = getelementptr inbounds { i32, i32, { double*, i32, i32 }, { double*, i32,
i32 }, { double*, i32, i32 }, { double*, i32, i32 } }, { i32, i32, { double*, i32,
```

```
i32 }, { double*, i32, i32 }, { double*, i32, i32 }, { double*, i32, i32 } }* %img,
i32 0, i32 2
  %116 = load { double*, i32, i32 }, { double*, i32, i32 }* %115
  %117 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %114, { double*, i32, i32 }* %117
  %118 = extractvalue { double*, i32, i32 } %114, 1
  %119 = extractvalue { double*, i32, i32 } %114, 2
  %120 = mul i32 %118, %119
  %121 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %116, { double*, i32, i32 }* %121
  %122 = extractvalue { double*, i32, i32 } %116, 1
  %123 = extractvalue { double*, i32, i32 } %116, 2
  %124 = extractvalue { double*, i32, i32 } %114, 1
  %125 = extractvalue { double*, i32, i32 } %114, 2
  %126 = extractvalue { double*, i32, i32 } %116, 1
  %127 = mul i32 %126, %125
  %128 = alloca { double*, i32, i32 }
  %129 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %128,
i32 0, i32 0
  %mallocsize = mul i32 %127, ptrtoint (double* getelementptr (double, double* null,
i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %130 = bitcast i8* %malloccall to double*
  %131 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %128,
i32 0, i32 1
  %132 = getelementptr inbounds { double*, i32, i32 }, { double*, i32, i32 }* %128,
i32 0, i32 2
  store double* %130, double** %129
  store i32 %126, i32* %131
  store i32 %125, i32* %132
  call void @_mat_mat_multiply({ double*, i32, i32 }* %117, { double*, i32, i32 }*
%121, { double*, i32, i32 }* %128)
  %133 = load { double*, i32, i32 }, { double*, i32, i32 }* %128
  %134 = alloca { double*, i32, i32 }
  store { double*, i32, i32 } %133, { double*, i32, i32 }* %134
  call void @_mat_print({ double*, i32, i32 }* %134)
  ret i32 0
}

declare noalias i8* @malloc(i32)
```

7 Lessons Learned

7.1 Team Members

Dimitri Borgers: I started this project with a very minimal understanding of how a language was compiled. Simply creating a scanner and parser was eye-opening and gave me a much better understanding of what the languages I have programmed with in the past were doing under the hood. Course knowledge aside, this was also my first time working on such a large project, with multiple team members. Coordinating among individuals is not as easy as it looks, and more importantly, making sure we all had the same vision of what our language should do and how it should do so was more work than we had originally thought. If there’s one thing I learned from this, it’s that you shouldn’t be afraid to ask the “dumb” questions.

Tahmid Munat: This is the first class where I learned functional programming and use the knowledge to implement a language that we imagined. Before the first week, I had little to no knowledge about the inner details, i.e. how a scanner, parser, ast etc. works. The deliverables were perfectly paced and by the end I was able to tie all the strings together and even compare how other traditional languages may have been created. All in all, this class was definitely very rewarding in terms of originality and content learned.

For the team project, coordinating between a group to find a common ground and then keeping at it for the whole semester was a valuable experience. All of us had different ideas and skill-sets and the regular meetings helped us to be on pace. The development tools used for the project were also beneficial. Overall, compared to other CS courses at Columbia, this group project turned out to be more demanding but also more rewarding.

Steven Bonilla: So far my experience with programming up to the point before taking this course had been only with traditional imperative languages like Java, Python, and C++. I had gotten my feet wet with “pseudo-functional” programming using a reactive framework in Java. But honestly that didn’t really prepare me for learning OCaml. I had to break down my mindset of programming linearly to learn functional programming. I had to think in recursion, and think more of how to deal with immutable variables and data structures.

Originally I didn’t enjoy programming in OCaml at all, but after about a couple weeks of struggling, I started to enjoy it and appreciate more nuanced features like a function returning a tuple of different types -- a convenient feature I wish Java had. Overall, this course set me in a good position to learn other functional languages and functional-style features of languages like in Scala.

Willie Koomson: I learned a lot about using functional programming practically in this course. I hadn’t used a functional language for anything more than theoretical exercises before this point. Additionally, I learned about the implementation of low-level image operations, through matrix operations. Most importantly, I've gained a solid understand of the architecture of compilers, which I think will make me a much better programmer.

Takuma Yasuda: Before this course, I hadn't learned any functional languages like OCaml. My first impression on OCaml was that this would not be my favorite language because it is quite different from languages I was used to such as C/C++, Java, and Python. However, after a while, I found it interesting to learn and program in OCaml because it is well-structured and more "mathematical" than C or Java. Thanks to this experience, I became interested in theoretical aspects of programming languages and their relations with mathematics. Also, I learned how compilers are organized and how they work through the project.

7.2 Advice for Future Teams

Have every member understand what each person is creating in their program. It is very easy for each individual to focus on one particular part of their language, and rely on the others to understand the rest. Unfortunately, if you choose to go down this path, you will soon realize that you have lost track of what your language can do as a whole and how exactly it implements its many tools. The team should work as one brain, not four or five individual machines that can't communicate. While it may take more time in the beginning to explain and learn the modules others have completed, it will make implementing the more demanding executables in the later parts of the project much easier.

8 Appendix

8.1 scanner.mll

Author: Tahmid Munat

```
(* Ocamllex scanner for Colode *)

{ open Parser }

let digit = ['0' - '9']
let digits = digit+
let char_lex = ['\x00' - '\x7F']
let string_lex = char_lex+
let strings = '"' ( [^ '\\' '"'] | '\\' [^ '\n'] ) * '"'

rule token = parse
  [' ' '\r' '\t' '\n'] { token lexbuf } (* Whitespace *)
| "//"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LBRACE }
| ']'      { RBRACE }
| '{'      { LBLOCK }
| '}'      { RBLOCK }
| ';'      { SEMI }
| '|'     { PIPE }
| ','      { COMMA }
| ':'      { COLON }
| '.'     { DOT }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '^'      { EXPONENT }
| '%'      { MODULUS }
| '~'     { TILDE }
| '='      { ASSIGN }
| "+="     { ASSIGNADD }
| "-="     { ASSIGNMINUS }
| "*="     { ASSIGNTIMES }
| "/="     { ASSIGNDIVIDE }
```

```
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "->"     { ARROW }
| "**"     { CONV }
| "and"    { AND }
| "or"     { OR }
| "not"    { NOT }
| "if"     { IF }
| "in"     { IN }
| "else"   { ELSE }
| "elif"   { ELIF }
| "for"    { FOR }
| "while"  { WHILE }
| "break"  { BREAK }
| "continue" { CONTINUE }
| "def"   { DEF }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "char"   { CHAR }
| "string" { STRING }
| "list"   { LIST }
| "void"   { VOID }
| "Image"  { IMAGE }
| "Pixel"  { PIXEL }
| "matrix" { MATRIX }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm {
FLIT(lxm) }
| strings as s { LITERALSTRING(s) }
| '\'' (char_lex as c) '\'' { LITERALCHAR(c) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
char)) }
```

```
and comment = parse
  '\n' { token lexbuf }
| _    { comment lexbuf }
```

8.2 parser.mly

Author: Dimitri Borgers

```
/* Ocamlyacc parser for Colode */

%{ open Ast %}

%token SEQUENCE LBLOCK RBLOCK LPAREN RPAREN LBRACE RBRACE SEMI COMMA
PLUS MINUS TIMES DIVIDE
%token EXPONENT MODULUS ASSIGN ASSIGNADD ASSIGNMINUS ASSIGNTIMES
ASSIGNDIVIDE NOT EQ NEQ LT LEQ GT GEQ AND OR DOT
%token RETURN IF ELSE ELIF FOR WHILE BREAK CONTINUE DEF INT BOOL
FLOAT VOID IN
%token CHAR STRING LIST IMAGE PIXEL MATRIX COLON CONV PIPE TILDE
POWER ARROW
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT LITERALSTRING
%token <char> LITERALCHAR
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE NOELIF ONED
%nonassoc ELSE ELIF  LBRACE LPAREN  RBRACE RPAREN
%right ASSIGN ASSIGNADD ASSIGNMINUS ASSIGNDIVIDE ASSIGNTIMES
%left OR DOT PIPE
%left AND
%left EQ NEQ POWER
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE LEFT
%left EXPONENT MODULUS CONV ID SEMI
%right NOT NEG TILDE
```

```
%%

program:
  decls EOF { let s, f = $1 in
    let s = List.rev s in (s, f) }

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

fdecl:
   DEF ID LPAREN formals_opt RPAREN COLON typ compound_stmt
     { { typ = $7;
       fname = $2;
       formals = $4;
       locals = [];
       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  }
  | CHAR  { Char }
  | typ LIST  { ArrayList($1) }
  | STRING { String }
  | IMAGE  { Image }
  | PIXEL  { Pixel }
  | MATRIX { Matrix }
/*
vdecl_list:
       { [] }
  | vdecl_list COMMA vdecl { $2 :: $1 }
```

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

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

compound_stmt:
 | LBLOCK stmt_list RBLOCK { $2 }

stmt:
   expr SEMI                           { Expr $1              }
  | RETURN expr_opt SEMI                          { Return $2
}
  | IF expr stmt %prec NOELSE { If($2, $3, Block([])) }
  | IF expr stmt ELIF expr stmt %prec NOELSE   { If($2, $3, If($5,
$6, Block([])))  }
  | IF expr stmt ELIF expr stmt ELSE stmt  { If($2, $3, If($5, $6,
$8))  }
  | IF expr stmt ELSE stmt  %prec NOELIF  { If($2, $3, $5)        }
  | FOR expr_opt SEMI expr SEMI expr_opt SEMI stmt
                                                 { For($2, $4, $6, $8)   }
  | WHILE expr stmt           { While($2, $3)         }
  | vdecl                                        { Declare(fst $1, snd $1)
}
  | compound_stmt {Block(List.rev $1)}

expr_opt:
  |  { Noexpr }
  | expr          { $1 }

/*member:
 DOT ID { [$2] }
 | member DOT ID { $3 :: $1 }
*/

array_index: ID DOT atom  { ArrayIndex(Id($1),$3) }

matrix_index:  ID DOT atom PIPE atom { Array2DIndex(Id($1), $3, $5)}

array_names:  array_index {$1}
    | matrix_index {$1}
```

```
image_index: ID ARROW ID { ImageIndex(Id($1), $3)}

name: ID {Id($1)}
    | array_names {$1}
    | image_index { $1 }

atom: LITERAL           { Literal($1)              }
  | FLIT       { Fliteral($1)             }
  | BLIT             { BoolLit($1)              }
  | LITERALCHAR      { CharLiteral($1)          }
  | LITERALSTRING    { StringLiteral(String.sub $1 1 ((String.length
$1)-2))          }
  | ID               { Id($1)                   }

term:  term PLUS   term { Binop($1, Add,   $3)     }
  | term MINUS  term { Binop($1, Sub,   $3)     }
  | term TIMES  term { Binop($1, Mult,  $3)     }
  | term EXPONENT  term { Binop($1, Exp,  $3)   }
  | term DIVIDE term { Binop($1, Div,   $3)     }
  | term EQ     term { Binop($1, Equal, $3)     }
  | term NEQ    term { Binop($1, Neq,   $3)     }
  | term LT     term { Binop($1, Less,  $3)     }
  | term LEQ    term { Binop($1, Leq,   $3)     }
  | term GT     term { Binop($1, Greater, $3)   }
  | term GEQ    term { Binop($1, Geq,   $3)     }
  | term AND    term { Binop($1, And,   $3)     }
  | term OR     term { Binop($1, Or,    $3)     }
  | term CONV     term { Binop($1, Conv,    $3) }
  | array_index  { $1 }
  | matrix_index { $1 }
  | image_index { $1 }
  | atom {$1}

expr:
  NOT expr         { Unop(Not, $2)            }
  | name ASSIGN expr   { Assign($1, $3)         }
  | typ ID ASSIGN expr   { DeclAssign($1, $2, $4)    }
  | name ASSIGNADD expr   { AssignAdd($1, $3)        }
  | name ASSIGNMINUS expr   { AssignMinus($1, $3)    }
  | name ASSIGNTIMES expr   { AssignTimes($1, $3)    }
  | name ASSIGNDIVIDE expr   { AssignDivide($1, $3) }
  | ID LPAREN args_opt RPAREN { Call($1, $3)         }
```

```
  /*| LPAREN expr RPAREN { $2 } */
  | array_lit          { $1 }
  | matrix_lit        { $1 }
  /*| atom LBRACE atom RBRACE LBRACE atom RBRACE {
Array2DIndex($1,$3, $6) }*/
  /*| ID member     { MemberAccess(Id($1), List.rev $2) }*/
  | term {$1}


array_lit: LBRACE array_opt RBRACE { Array(List.rev $2) }

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

matrix_lit: TILDE LBRACE rows RBRACE { Array2D(List.rev $3)}

rows:
 row { [List.rev $1] }
 | rows PIPE row { $3 :: (List.rev $1) }

row:
    LITERAL { [Fliteral(string_of_int $1)] }
  | FLIT { [Fliteral($1)]}
  | row FLIT { Fliteral($2) :: $1 }
  | row LITERAL { Fliteral(string_of_int $2) :: $1 }

args_opt:
    /* nothing */ { [] }
  | args_list  { List.rev $1 }

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

8.3 ast.ml

Author: Steven Bonilla

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

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

type uop = Neg | Not

type typ = Int | Bool | Float | Void | Char | ArrayList of typ |
String | Image | Pixel | Matrix

type bind = typ * string

type expr =
     Literal of int
   | Fliteral of string
   | BoolLit of bool
   | CharLiteral of char
   | StringLiteral of string
   | Id of string
   | Binop of expr * op * expr
   | Unop of uop * expr
   | Assign of expr * expr
   | AssignAdd of expr * expr
   | AssignMinus of expr * expr
   | AssignTimes of expr * expr
   | AssignDivide of expr * expr
   | DeclAssign of typ * string * expr
   | Call of string * expr list
   | Array of expr list
   | ArrayIndex of expr * expr
   | Array2D of expr list list
   | Array2DIndex of expr * expr * expr
   | ImageIndex of expr * string
   | MemberAccess of expr * string list
   | 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
   | Declare of typ * string
```

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

type program = stmt 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 -> "**"
  | Exp -> "^"

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

let rec string_of_typ = function
      Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | Char -> "char"
  | ArrayList(t) -> string_of_typ t ^ "list"
  | String -> "string"
  | Image -> "image"
  | Pixel -> "pixel"
```

```
  | Matrix -> "matrix"

let rec string_of_expr = function
     Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | 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) -> string_of_expr v ^ " = " ^ string_of_expr e
  | AssignAdd(v, e) -> string_of_expr v ^ " += " ^ string_of_expr e
  | AssignMinus(v, e) -> string_of_expr v ^ " -= " ^ string_of_expr e
  | AssignTimes(v, e) -> string_of_expr v ^ " *= " ^ string_of_expr e
  | AssignDivide(v, e) -> string_of_expr v ^ " /= " ^ string_of_expr
e
  | DeclAssign(t, v, e) -> (string_of_typ t) ^ v ^ " /= " ^
string_of_expr e
  | Call(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Array(l) -> "[" ^ (String.concat ", " (List.map string_of_expr
l)) ^ "]"
  | ArrayIndex(a, b) -> string_of_expr a ^ "[" ^ string_of_expr b ^
"]"
  | Array2DIndex(a, b, c) -> string_of_expr a ^ "[" ^ string_of_expr
b ^ "]" ^ "[" ^ string_of_expr c ^ "]"
  | MemberAccess(a, b) -> string_of_expr a ^ "." ^String.concat "." b
  | CharLiteral(c) -> "'" ^ Char.escaped c ^ "'"
  | StringLiteral(s) -> "\"" ^ s ^ "\""
  | Array2D(l) -> "~[" ^ (String.concat "|"
     (List.map (fun row -> String.concat " " (List.map
string_of_expr row))
     l))
  ^ "]"
  | ImageIndex(id, chan) -> string_of_expr id ^ "->" ^ chan
  | 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, st) ->
     "for " ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; "
^ string_of_expr e3 ^" "^ string_of_stmt st
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
string_of_stmt s
  | Declare(t, s) -> (string_of_typ t) ^ " " ^ s


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 (stmts, funcs) =
  String.concat "" (List.map string_of_stmt stmts) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.4 codegen.ml

Author: Willie Koomson

```
module L = Llvm
module A = Ast
module M = Matrix
```

```
open Sast

module StringMap = Map.Make(String)
let translate (statements, functions) =
    let make_err e = raise (Failure e) in
    let context = L.global_context () in
    (* Primitive types *)
    let i32_t = L.i32_type context
      and i8_t = L.i8_type context
      and i1_t = L.i1_type  context (* used to represent boolean
type *)
      and float_t = L.double_type context
      and void_t = L.void_type context
      and char_t = L.i8_type context
    in
    (* Compound types *)
    let list_t = fun (inner_typ: L.lltype) -> L.struct_type context
[| L.pointer_type inner_typ; i32_t (*length*); i32_t (*capacity*)|]
in
    let string_t = L.struct_type context [| L.pointer_type char_t;
i32_t (*length*); |] in
    let matrix_t = L.struct_type context [| L.pointer_type float_t;
i32_t (*width*); i32_t (*height*) |]  in
    let image_t  = L.struct_type context [| i32_t (*width*); i32_t (*
height *); matrix_t; matrix_t; matrix_t; matrix_t; |] in
    let pixel_t = L.vector_type float_t 4 in
    (* Internal constants *)
    let zero = L.const_int i32_t 0 in
    let one = L.const_int i32_t 1 in
    let true_ = L.const_int i1_t 1 in
    let false_ = L.const_int i1_t 0 in
    let const_i32_of = L.const_int (L.i32_type context) in
    let const_float_of = L.const_float float_t in
    (* Main module *)
    let code_module = L.create_module context "Colode" in
    let rec ltype_of_typ = function
        A.Int   -> i32_t
      | A.Bool  -> i1_t
      | A.Float -> float_t
      | A.Void  -> void_t
      | A.Char -> char_t
      | A.ArrayList t -> list_t (ltype_of_typ t)
      | A.String -> string_t
```

```
      | A.Image -> image_t
      | A.Matrix -> matrix_t
      | A.Pixel -> pixel_t
    in
    let print_t = L.function_type i32_t [| L.pointer_type char_t |]
in
    let print_func = L.declare_function "puts" print_t code_module in
    let pow_t = L.function_type float_t [| float_t; float_t |] in
    let pow_func = L.declare_function "pow" pow_t code_module in
    let printf_t = L.var_arg_function_type i32_t [| (L.pointer_type
char_t)|] in
    let printf_func = L.declare_function "printf" printf_t
code_module in
    let mat_zero_out_t = L.function_type void_t [| L.pointer_type
matrix_t |] in
    let mat_zero_out_func = L.declare_function "_mat_zero_out"
mat_zero_out_t code_module in
    let mat_print_t = L.function_type void_t [|  L.pointer_type
matrix_t |] in
    let mat_print_func = L.declare_function "_mat_print" mat_print_t
code_module in
    let mat_scalar_t = L.function_type void_t [| L.pointer_type
matrix_t; float_t; L.pointer_type matrix_t |] in
    let mat_scalar_add_func = L.declare_function "_mat_scalar_add"
mat_scalar_t code_module in
    let mat_scalar_subtract_func = L.declare_function
"_mat_scalar_subtract" mat_scalar_t code_module in
    let mat_scalar_multiply_func = L.declare_function
"_mat_scalar_multiply" mat_scalar_t code_module in
    let mat_scalar_divide_func = L.declare_function
"_mat_scalar_divide" mat_scalar_t code_module in
    let mat_mat_t = L.function_type void_t [| L.pointer_type
matrix_t; L.pointer_type matrix_t; L.pointer_type matrix_t;|] in
    let mat_mat_add_func = L.declare_function "_mat_mat_add"
mat_mat_t code_module in
    let mat_mat_subtract_func = L.declare_function
"_mat_mat_subtract" mat_mat_t code_module in
    let mat_mat_multiply_func = L.declare_function
"_mat_mat_multiply" mat_mat_t code_module in
    let mat_mat_divide_func = L.declare_function "_mat_mat_divide"
mat_mat_t code_module in
    let mat_mat_convolute_func = L.declare_function
"_mat_mat_convolute" mat_mat_t code_module in
```

```
    let mat_equal_t = L.function_type i1_t [| L.pointer_type
matrix_t; L.pointer_type matrix_t|] in
    let mat_mat_equal_func = L.declare_function "_mat_mat_equal"
mat_equal_t code_module in
    let mat_power_t = L.function_type void_t [|L.pointer_type
matrix_t; i32_t; L.pointer_type matrix_t |] in
    let mat_mat_power_func = L.declare_function "_mat_mat_power"
mat_power_t code_module in
    let image_fn_t = L.function_type void_t [| L.pointer_type char_t;
L.pointer_type image_t |] in
    let image_read_func = L.declare_function "_image_read" image_fn_t
code_module in
    let image_write_func = L.declare_function "_image_write"
image_fn_t code_module in
    let mat_gauss_t = L.function_type void_t [| float_t;
L.pointer_type matrix_t |] in
    let mat_gen_gauss = L.declare_function "_mat_gen_gauss"
mat_gauss_t code_module in
    let mat_gen_sharpen = L.declare_function "_mat_gen_sharpen"
mat_zero_out_t code_module in
    let mat_gen_edge_detect = L.declare_function
"_mat_gen_edge_detect" mat_zero_out_t code_module in
    let mat_gen_brighten = L.declare_function "_mat_gen_brighten"
mat_gauss_t code_module in
    let function_decls =
      let func_decl map fd =
           let name = fd.sfname in
           let formal_types = Array.of_list (List.map (fun (t,_) ->
ltype_of_typ t) fd.sformals) in
           let func_type = L.function_type (ltype_of_typ fd.styp)
formal_types in
           StringMap.add name (L.define_function name func_type
code_module, fd) map
      in
      List.fold_left func_decl StringMap.empty functions
    in
    let add_terminal builder fn = match L.block_terminator
(L.insertion_block builder) with
        Some _ -> ()
      | None -> ignore (fn builder)
    in
    let lookups map name : L.llvalue = match StringMap.find_opt name
map    with
```

```
      Some v -> v | None -> make_err ("Couldn't find " ^ name)
     in
     let binop_char_concat lv rv name builder : L.llvalue = (* char
+ char = new string *)
      let alloc = L.build_alloca string_t name builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let len_loc = L.build_struct_gep alloc 1 "" builder in
      let len = const_i32_of 2 in
      let data_loc = L.build_array_alloca char_t len "" builder in
      let fst_loc = L.build_gep data_loc [|zero; const_i32_of 0 |]
"" builder in
      let snd_loc = L.build_gep data_loc [|zero; const_i32_of 1 |]
"" builder in
      let _ = L.build_store lv fst_loc builder in
      let _ = L.build_store rv snd_loc builder in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store len len_loc builder
      in alloc
    in
    let genIf (builder: L.llbuilder) (this : L.llvalue) (pred:
L.llvalue) (then_s: L.llbuilder -> L.llbuilder)
      (else_s: L.llbuilder -> L.llbuilder)
      : L.llbuilder =
      let merge_bb = L.append_block context "if_merge" this in
      let branch_ins = L.build_br merge_bb in
      let then_bb = L.append_block context "if_then" this in
      let then_builder = then_s (L.builder_at_end context then_bb)
in
      let () = add_terminal then_builder branch_ins in
      let else_bb = L.append_block context "if_else" this in
      let else_builder = else_s (L.builder_at_end context else_bb)
in
      let () = add_terminal else_builder branch_ins in
      let _ = L.build_cond_br pred then_bb else_bb builder in
      (L.builder_at_end context merge_bb)
    in
    let genWhile (builder: L.llbuilder) (this : L.llvalue) (pred:
L.llbuilder -> L.llvalue * L.llbuilder) (body: L.llbuilder ->
L.llbuilder)
      : L.llbuilder =
      let pred_bb = L.append_block context "while" this in
      let _ = L.build_br pred_bb builder in
      let body_bb = L.append_block context "while_body" this in
```

```
        let body_bldr = body (L.builder_at_end context body_bb) in
        let () = add_terminal body_bldr (L.build_br pred_bb) in
        let pred_bldr = L.builder_at_end context pred_bb in
        let bool_val, pred_bldr = pred pred_bldr in
        let merge_bb = L.append_block context "while_merge" this in
        let _ = L.build_cond_br bool_val body_bb merge_bb pred_bldr in
        (L.builder_at_end context merge_bb)    in
    let binop_array_concat ty (this: L.llvalue) (*Llvm func def*) lv
rv name builder : L.llvalue * L.llbuilder = (* array + array = new
array *)
        let l_type = ltype_of_typ ty in
        let alloc = L.build_alloca (list_t l_type) name builder in
        let data_field_loc = L.build_struct_gep alloc 0 "" builder in
        let len_loc = L.build_struct_gep alloc 1 "" builder in
        let cap_loc = L.build_struct_gep alloc 2 "" builder in
        (* let ldata_field_loc = L.build_struct_gep lv 0 "" builder in
*)
        (* let llen_loc = L.build_struct_gep lv 1 "" builder in *)
        (* let rdata_field_loc = L.build_struct_gep rv 0 "" builder in
*)
        (* let rlen_loc = L.build_struct_gep rv 1 "" builder in *)
        let ldata_loc = L.build_extractvalue lv 0 "" builder in
        let llen = L.build_extractvalue lv 1 "" builder in
        let rdata_loc = L.build_extractvalue rv 0 "" builder in
        let rlen = L.build_extractvalue rv 1 "" builder in
        let len = L.build_add llen rlen "" builder in
        (* let () = L.print_module "codegen.out" code_module in
        let () = L.dump_module code_module in *)
        let cap = L.build_mul len (const_i32_of 2) "" builder in
        let pred = L.build_icmp L.Icmp.Eq cap zero "" builder in
        let builder = genIf builder this pred
              (*Then*)
              (fun b -> let _ = L.build_store (L.const_bitcast
(L.const_pointer_null l_type) (L.pointer_type l_type)) data_field_loc
b in b )
              (*Else*)
              (fun b -> let alloc = L.build_array_alloca l_type cap ""
builder in
                    let _ = L.build_store alloc data_field_loc b in b )
        in
        let data_loc = L.build_load data_field_loc "" builder in
        let iter = L.build_alloca i32_t "iter" builder in
        let _ = L.build_store zero iter builder in
```

```
      let builder = genWhile builder this
            (*pred*)
            (fun b -> let i = L.build_load iter "" b in
                  (L.build_icmp L.Icmp.Slt i len "" b, b) )
            (*body*)
            (fun b ->
                  let i = L.build_load iter "" b in
                  let use_left = L.build_icmp L.Icmp.Slt i llen "" b
in
                  let lgep = L.build_gep ldata_loc [| i |] "" b in
                  let rindex = L.build_sub i llen "" b in
                  let rgep = L.build_gep rdata_loc [| rindex |] "" b
in
                  let gep = L.build_select use_left lgep rgep "" b in
                  let value = L.build_load gep "" b in
                  let new_addr = L.build_gep data_loc [|i|] "" b in
                  let _ = L.build_store value new_addr b in
                  let incr = L.build_add i one "" b in
                  let _ = L.build_store incr iter b in
            b)
      in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store len len_loc builder in
      let _ = L.build_store cap cap_loc builder in
      let value = L.build_load alloc "" builder in
      value, builder
    in
    let binop_str_concat (this: L.llvalue) (*Llvm func def*) lv rv
name builder : L.llvalue * L.llbuilder =
      let alloc = L.build_alloca string_t name builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let len_loc = L.build_struct_gep alloc 1 "" builder in
      (* let ldata_field_loc = L.build_struct_gep lv 0 "" builder in
*)
      let ldata_loc = L.build_extractvalue lv 0 "" builder in
      (* let llen_loc = L.build_struct_gep lv 1 "" builder in *)
      let llen = L.build_extractvalue lv 1 "" builder in
      (* let rdata_field_loc = L.build_struct_gep rv 0 "" builder in
*)
      let rdata_loc = L.build_extractvalue rv 0 "" builder in
      (* let rlen_loc = L.build_struct_gep rv 1 "" builder in *)
      let rlen = L.build_extractvalue rv 1 "" builder in
      let len = L.build_add llen rlen "" builder in
```

```
      let pred = L.build_icmp L.Icmp.Eq len zero "" builder in
      let builder = genIf builder this pred
            (*Then*)
            (fun b -> let _ = L.build_store (L.const_bitcast
(L.const_pointer_null i8_t) (L.pointer_type i8_t)) data_field_loc b
in b )
            (*Else*)
            (fun b -> let alloc = L.build_array_alloca i8_t len ""
builder in
                  let _ = L.build_store alloc data_field_loc b in b )
      in
      let data_loc = L.build_load data_field_loc "" builder in
      let iter = L.build_alloca i32_t "iter" builder in
      let _ = L.build_store zero iter builder in
      let builder = genWhile builder this
            (*pred*)
            (fun b -> let i = L.build_load iter "" b in
                  (L.build_icmp L.Icmp.Slt i len "" b, b) )
            (*body*)
            (fun b ->
                  let i = L.build_load iter "" b in
                  let use_left = L.build_icmp L.Icmp.Slt i llen "" b
in
                  let lgep = L.build_gep ldata_loc [|i |] "" b in
                  let rindex = L.build_sub i llen "" b in
                  let rgep = L.build_gep rdata_loc [| rindex |] "" b
in
                  let gep = L.build_select use_left lgep rgep "" b in
                  let value = L.build_load gep "" b in
                  let new_addr = L.build_gep data_loc [|i|] "" b in
                  let _ = L.build_store value new_addr b in
                  let incr = L.build_add i one "" b in
                  let _ = L.build_store incr iter b in
            b )
      in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store len len_loc builder in
      let value = L.build_load alloc "" builder in
      value, builder
    in
    let binop_str_equal (this: L.llvalue) (*Llvm func def*) lv rv
name builder : L.llvalue * L.llbuilder =
      let ldata_loc = L.build_extractvalue lv 0 "" builder in
```

```
      let llen = L.build_extractvalue lv 1 "" builder in
      let rdata_loc = L.build_extractvalue rv 0 "" builder in
      let rlen = L.build_extractvalue rv 1 "" builder in
      let pred = L.build_icmp L.Icmp.Ne rlen llen "" builder in
      let is_equal = L.build_alloca i1_t "is_equal" builder in
      let iter = L.build_alloca i32_t "iter" builder in
      let _ = L.build_store zero iter builder in
      let _ = L.build_store true_ is_equal builder in
      let builder = genIf builder this pred
           (*Then*)
           (fun b -> let _ = L.build_store false_ is_equal b in b )
           (*Else*)
           (fun b -> genWhile b this
                (*pred*)
                (fun b -> let i = L.build_load iter "" b in
                     (L.build_icmp L.Icmp.Slt i llen "" b, b) )
                (*body*)
                (fun b -> let i = L.build_load iter "" b in
                     let litem_loc = L.build_gep ldata_loc [| i |]
"" b in
                     let ritem_loc = L.build_gep rdata_loc [| i |]
"" b in
                     let lchar = L.build_load litem_loc "" b in
                     let rchar = L.build_load ritem_loc "" b in
                     let pred = L.build_icmp L.Icmp.Ne rchar lchar
"" b in
                     let b = genIf b this pred
                          (*then*)
                          (fun b -> let _ = L.build_store false_
is_equal b in b)
                          (fun b -> b)
                     in
                     let incr = L.build_add i one "" b in
                     let _ = L.build_store incr iter b in  b )
           )
      in
      let eq = L.build_load is_equal "" builder in
      (eq, builder)
    in
    let const_char c = L.const_int i8_t (Char.code c) in
     let rec expr map builder (this: L.llvalue) (*Llvm func def*)
(typ, sx) : (L.llvalue * L.llvalue StringMap.t * L.llbuilder) =
     match sx with
```

```
      SLiteral i -> (L.const_int i32_t i, map, builder)
    | SBoolLit b -> (L.const_int i1_t (if b then 1 else 0), map,
builder)
    | SFliteral l -> (L.const_float_of_string float_t l, map,
builder)
    | SCharLiteral c -> (const_char c, map, builder)
    | SStringLiteral s -> let alloc = L.build_alloca string_t ""
builder in (* eventually figure out a way to store value in registers
instead of making an extra allocation*)
      let str_global = L.build_global_string s "" builder in
      let str = L.build_bitcast str_global (L.pointer_type i8_t) ""
builder in
      let str_field_loc = L.build_struct_gep alloc 0 "" builder in
      let str_len = L.const_int i32_t (String.length s) in
      let len_loc = L.build_struct_gep alloc 1 "" builder in
      let _ = L.build_store str str_field_loc builder in
      let _ = L.build_store str_len len_loc builder in
      let value = L.build_load alloc "" builder
    in (value, map, builder)
    | SNoexpr -> (L.const_int i32_t 0, map, builder)
    | SId s -> (L.build_load (lookups map s) s builder, map, builder)
    | SCall ("print", [ex]) -> let s_lval, _, builder = expr map
builder this ex in
      (* let s = L.build_struct_gep s_lval 0 "" builder in *)
      let s = L.build_extractvalue s_lval 0 "" builder in
      (* let olen = L.build_extractvalue s_lval 1 "" builder in
      let len = L.build_add olen one "" builder in
      let stringz = L.build_array_alloca i8_t len "" builder in
//*alloc space for null-terminated string
      let _ = L.build_store s stringz builder in
      let terminal = L.build_gep stringz [| olen |] "" builder in
      let _ = L.build_store zero terminal in *)
      (L.build_call print_func [|s|] "" builder, map, builder)
    | SCall ("iprint", [ex]) -> let s_lval, _, builder = expr map
builder this ex in
      let decimal_spec = L.build_global_stringptr "%d" "" builder in
      (L.build_call printf_func [|decimal_spec; s_lval|] "" builder,
map, builder)
    | SCall ("fprint", [ex]) -> let s_lval, _, builder = expr map
builder this ex in
      (* let decimal_spec = L.build_array_alloca i8_t (const_i32_of
2) "" builder in
```

```
      let fst_idx = L.build_gep decimal_spec [| zero |] "" builder
in
      let snd_idx = L.build_gep decimal_spec [| one |] "" builder in
      let _ = L.build_store (const_char '%') fst_idx builder in
      let _ = L.build_store (const_char 'f') snd_idx builder in *)
      let decimal_spec = L.build_global_stringptr "%f" "" builder in
      (L.build_call printf_func [|decimal_spec; s_lval|] "" builder,
map, builder)
    | SCall ("mprint", [ex]) ->
      let arg_v, _, builder = expr map builder this ex in
      let arg_p = L.build_alloca matrix_t "" builder in
      let _ = L.build_store arg_v arg_p builder in
      (L.build_call mat_print_func [|arg_p|] "" builder, map,
builder)
    | SCall ("new", [widthx; heightx]) ->
      let width_v, _, builder = expr map builder this widthx in
      let height_v, _, builder = expr map builder this heightx in
      let size, builder = M.llvm_mat_size width_v height_v ""
builder in
      let alloc = L.build_alloca matrix_t "" builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let width_loc = L.build_struct_gep alloc 1 "" builder in
      let height_loc = L.build_struct_gep alloc 2 "" builder in
      let data_loc = L.build_array_alloca float_t size "" builder in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store width_v width_loc builder in
      let _ = L.build_store height_v height_loc builder in
      let _ = L.build_call mat_zero_out_func [| alloc |] "" builder
in
      let value = L.build_load alloc "" builder in
      (value, map, builder)
    | SCall ("generate_gaussian", [widthx; heightx; sigmax]) ->
      let width_v, _, builder = expr map builder this widthx in
      let height_v, _, builder = expr map builder this heightx in
      let sigma_v, _, builder = expr map builder this sigmax in
      let size, builder = M.llvm_mat_size width_v height_v ""
builder in
      let alloc = L.build_alloca matrix_t "" builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let width_loc = L.build_struct_gep alloc 1 "" builder in
      let height_loc = L.build_struct_gep alloc 2 "" builder in
      let data_loc = L.build_array_alloca float_t size "" builder in
      let _ = L.build_store data_loc data_field_loc builder in
```

```
      let _ = L.build_store width_v width_loc builder in
      let _ = L.build_store height_v height_loc builder in
      let _ = L.build_call mat_gen_gauss [| sigma_v; alloc |] ""
builder in
      let value = L.build_load alloc "" builder in
      (value, map, builder)
    | SCall ("generate_sharpen", []) ->
      let width_v = const_i32_of 3 in
      let height_v = const_i32_of 3 in
      let size, builder = M.llvm_mat_size width_v height_v ""
builder in
      let alloc = L.build_alloca matrix_t "" builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let width_loc = L.build_struct_gep alloc 1 "" builder in
      let height_loc = L.build_struct_gep alloc 2 "" builder in
      let data_loc = L.build_array_alloca float_t size "" builder in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store width_v width_loc builder in
      let _ = L.build_store height_v height_loc builder in
      let _ = L.build_call mat_gen_sharpen [| alloc |] "" builder in
      let value = L.build_load alloc "" builder in
      (value, map, builder)
    | SCall ("generate_edge_detect", []) ->
      let width_v = const_i32_of 3 in
      let height_v = const_i32_of 3 in
      let size, builder = M.llvm_mat_size width_v height_v ""
builder in
      let alloc = L.build_alloca matrix_t "" builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let width_loc = L.build_struct_gep alloc 1 "" builder in
      let height_loc = L.build_struct_gep alloc 2 "" builder in
      let data_loc = L.build_array_alloca float_t size "" builder in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store width_v width_loc builder in
      let _ = L.build_store height_v height_loc builder in
      let _ = L.build_call mat_gen_edge_detect [| alloc |] ""
builder in
      let value = L.build_load alloc "" builder in
      (value, map, builder)
    | SCall ("generate_brighten", [ex]) ->
      let intensity, _, builder = expr map builder this ex in
      let width_v = const_i32_of 3 in
      let height_v = const_i32_of 3 in
```

```
      let size, builder = M.llvm_mat_size width_v height_v ""
builder in
      let alloc = L.build_alloca matrix_t "" builder in
      let data_field_loc = L.build_struct_gep alloc 0 "" builder in
      let width_loc = L.build_struct_gep alloc 1 "" builder in
      let height_loc = L.build_struct_gep alloc 2 "" builder in
      let data_loc = L.build_array_alloca float_t size "" builder in
      let _ = L.build_store data_loc data_field_loc builder in
      let _ = L.build_store width_v width_loc builder in
      let _ = L.build_store height_v height_loc builder in
      let _ = L.build_call mat_gen_brighten [| intensity; alloc |]
"" builder in
      let value = L.build_load alloc "" builder in
      (value, map, builder)
    | SCall ("coload", [ex]) ->
      let s_lval, _, builder = expr map builder this ex in
      let s = L.build_extractvalue s_lval 0 "" builder in
      let alloc = L.build_alloca image_t "" builder in
      let _ = L.build_call image_read_func [| s; alloc |] "" builder
in
      let value = L.build_load alloc "" builder in
      (value, map, builder)
    | SCall ("coclose", [imgx; filename]) ->
      let s_lval, _, builder = expr map builder this filename in
      let img_str, _, builder = expr map builder this imgx in
      let name_s = L.build_extractvalue s_lval 0 "" builder in
      let alloc = L.build_alloca image_t "" builder in
      let _ = L.build_store img_str alloc builder in
      let _ = L.build_call image_write_func [| name_s; alloc |] ""
builder in
      (zero, map, builder)
    (*Add rest of built-in functions here *)
    | SCall (name, exl) -> let (ldef, fd) = StringMap.find name
function_decls in
      let args = List.map (fun (a,b,c) -> a) (List.rev (List.map
(expr map builder this) (List.rev exl))) in
      let call = L.build_call ldef (Array.of_list args) "" builder
in
      (call, map, builder)
    | SAssign(lex, rex) -> let rval, m', builder = expr map builder
this rex in
      (match (snd lex) with
        SId s -> let addr = lookups map s in
```

```
            let _ = L.build_store rval addr builder in
            (rval, m', builder)
        | SArrayIndex(id, idx) -> let name = match snd id with
                SId s -> s
                | _ -> "err:cannot index non-id"
            in
            let a_addr = lookups map name in
            let data_field_loc = L.build_struct_gep a_addr 0 ""
builder in
            let data_loc = L.build_load data_field_loc "" builder in
            let ival, _, builder = expr map builder this idx in
            let addr = L.build_gep data_loc [| zero; ival |] ""
builder  in
            let _ = L.build_store rval addr builder in
            (rval, m', builder)
        | SArray2DIndex(id, idx, idx2) -> let name = match snd id
with
                SId s -> s
                | _ -> "err:cannot index non-id"
            in
            let a_addr = lookups map name in
            let data_field_loc = L.build_struct_gep a_addr 0 ""
builder in
            let data_loc = L.build_load data_field_loc "" builder in
            let ival, _, builder = expr map builder this idx in
            let jval, _, builder = expr map builder this idx2 in
            let index, builder = M.llvm_mat_index a_addr ival jval ""
builder in
            let addr = L.build_gep data_loc [| index |] "" builder in
            let _ = L.build_store rval addr builder in
            (rval, m', builder)
        | SImageIndex(id, chan) ->
            let name = match snd id with
           SId s -> s
           | _ -> "err:cannot index non-id"
            in
            let img_addr = lookups map name in
            let index = match chan with "red" -> 2 | "green" -> 3 |
"blue" -> 4 | "alpha" -> 5 | _ -> 6  in
            let mat_loc = L.build_struct_gep img_addr index ""
builder in
            let _ = L.build_store rval mat_loc builder in
            (rval, m', builder)
```

```
        | _ -> make_err "Cannot assign to a non-name type. This
error should be caught by semantic checker."
       )
     | SDeclAssign(ty, s, rex) -> let l_type = ltype_of_typ ty in
       let addr = L.build_alloca l_type s builder in
       let rval, m', builder = expr map builder this rex in
       let m'' = StringMap.add s addr m' in
       let _ = L.build_store rval addr builder in
       (rval, m'', builder)
     | SArray sl -> let l_type = ltype_of_typ (match sl with [] ->
A.Void | _ -> (fst (List.hd sl)) ) in
       let ty = list_t l_type in
       let alloc = L.build_alloca ty "" builder in
       let data_field_loc = L.build_struct_gep alloc 0 "" builder in
       let len_loc = L.build_struct_gep alloc 1 "" builder in
       let cap_loc = L.build_struct_gep alloc 2 "" builder in
       let len = List.length sl in
       let cap = len * 2 in
       let data_loc = match cap with 0 -> L.const_pointer_null l_type
            | _ -> L.build_array_alloca l_type (const_i32_of cap) ""
builder
       in
       let sto (acc, builder) ex =
            let value, m', builder = expr map builder this ex in
            let item_loc = L.build_gep data_loc [|const_i32_of acc |]
"" builder in
            let _ = L.build_store value item_loc builder in
            (acc + 1, builder)
       in
       let _, builder = List.fold_left sto (0, builder) sl in
       let _ = L.build_store data_loc data_field_loc builder in
       let _ = L.build_store (const_i32_of len) len_loc builder in
       let _ = L.build_store (const_i32_of cap) cap_loc builder in
       let value = L.build_load alloc "" builder in
       (value, map, builder)
     | SArrayIndex(id, idx) ->
       let name = match snd id with
           SId s -> s
           | _ -> "err:cannot index non-id"
       in
       let a_addr = lookups map name in
       let data_field_loc = L.build_struct_gep a_addr 0 "" builder in
       let data_loc = L.build_load data_field_loc "" builder in
```

```
        let ival, _, builder = expr map builder this idx in
        let i_addr = L.build_gep data_loc [| ival |] "" builder in
        let value = L.build_load i_addr "" builder in
        (value, map, builder)
     | SArray2D sl ->
        let ty = matrix_t in
        let alloc = L.build_alloca ty "" builder in
        let data_field_loc = L.build_struct_gep alloc 0 "" builder in
        let width_loc = L.build_struct_gep alloc 1 "" builder in
        let height_loc = L.build_struct_gep alloc 2 "" builder in
        let width = List.length (List.hd sl) in
        let height = List.length sl in
        let size = (const_i32_of (M.mat_size width height)) in
        let data_loc = L.build_array_alloca float_t size "" builder
        in
        let row_store (i, builder) exl =
              let column_store (j, builder) ex =
                    let value, m', builder = expr map builder this ex in
                    let index = M.mat_index width i j in
                    let item_loc = L.build_gep data_loc [|const_i32_of
index |] "" builder in
                    let _ = L.build_store value item_loc builder in
                    (j + 1, builder)
              in
              let _, builder = List.fold_left column_store (0, builder)
exl in
              (i+1, builder)
        in
        let _, builder = List.fold_left row_store (0, builder) sl in
        let _ = L.build_store data_loc data_field_loc builder in
        let _ = L.build_store (const_i32_of width) width_loc builder
in
        let _ = L.build_store (const_i32_of height) height_loc builder
in
        let value = L.build_load alloc "" builder in
        (value, map, builder)
     | SArray2DIndex(id, idx, idx2) ->
        let name = match snd id with
             SId s -> s
             | _ -> "err:cannot index non-id"
        in
        let a_addr = lookups map name in
        let data_field_loc = L.build_struct_gep a_addr 0 "" builder in
```

```
      let data_loc = L.build_load data_field_loc "" builder in
      let ival, _, builder = expr map builder this idx in
      let jval, _, builder = expr map builder this idx2 in
      let index, builder = M.llvm_mat_index a_addr ival jval ""
builder in
      let i_addr = L.build_gep data_loc [| index |] "" builder in
      let value = L.build_load i_addr "" builder in
      (value, map, builder)
    | SImageIndex(id, chan) ->
      let name = match snd id with
          SId s -> s
          | _ -> "err:cannot index non-id"
      in
      let img_addr = lookups map name in
      let index = match chan with "red" -> 2 | "green" -> 3 | "blue"
-> 4 | "alpha" -> 5 | _ -> 6  in
      let mat_loc = L.build_struct_gep img_addr index "" builder in
      let mat = L.build_load mat_loc "" builder in
      (mat, map, builder)
    | SBinop(lex, op, rex) ->
      let lval, m', builder = expr map builder this lex in
      let rval, m'', builder = expr m' builder this rex in
      let ty = fst lex in
      (match ty with
        A.Int ->
           (match op with
                A.Add -> L.build_add lval rval "" builder, m'',
builder
                | A.Sub -> L.build_sub lval rval "" builder, m'',
builder
                | A.Mult -> L.build_mul lval rval "" builder, m'',
builder
                | A.Div -> L.build_sdiv  lval rval "" builder, m'',
builder
                | A.Equal -> L.build_icmp L.Icmp.Eq lval rval ""
builder, m'', builder
                | A.Neq -> L.build_icmp L.Icmp.Ne lval rval ""
builder, m'', builder
                | A.Less -> L.build_icmp L.Icmp.Slt lval rval ""
builder, m'', builder
                | A.Leq -> L.build_icmp L.Icmp.Sle lval rval ""
builder, m'', builder
```

```
                 | A.Greater -> L.build_icmp L.Icmp.Sgt lval rval ""
builder, m'', builder
                 | A.Geq -> L.build_icmp L.Icmp.Sge lval rval ""
builder, m'', builder
                 | A.And -> L.build_and lval rval "" builder, m'',
builder
                 | A.Or -> L.build_or lval rval "" builder, m'',
builder
                 | A.Exp -> let lfval = L.build_sitofp lval float_t
"" builder in
                      let rfval = L.build_sitofp rval float_t ""
builder in
                      let f_result = L.build_call pow_func [|lfval;
rfval|] "" builder in
                      let add_half = L.build_fadd f_result
(const_float_of 0.5) "" builder in
                      (L.build_fptosi add_half i32_t "" builder,
m'', builder)
                 | A.Conv -> make_err "internal error, cannot perform
this operation on integers"
            )
        | A.Float ->
            (match op with
                 A.Add -> L.build_fadd lval rval "" builder, m'',
builder
                 | A.Sub -> L.build_fsub lval rval "" builder, m'',
builder
                 | A.Mult -> L.build_fmul lval rval "" builder, m'',
builder
                 | A.Div -> L.build_fdiv lval rval "" builder, m'',
builder
                 | A.Equal -> L.build_fcmp L.Fcmp.Oeq lval rval ""
builder, m'', builder
                 | A.Neq -> L.build_fcmp L.Fcmp.One lval rval ""
builder, m'', builder
                 | A.Less -> L.build_fcmp L.Fcmp.Olt lval rval ""
builder, m'', builder
                 | A.Leq -> L.build_fcmp L.Fcmp.Ole lval rval ""
builder, m'', builder
                 | A.Greater -> L.build_fcmp L.Fcmp.Ogt lval rval ""
builder, m'', builder
                 | A.Geq -> L.build_fcmp L.Fcmp.Oge lval rval ""
builder, m'', builder
```

```
                 | A.Exp -> L.build_call pow_func [|lval; rval|] ""
builder, m'', builder
                 | _ -> make_err "internal error, cannot perform this
operation on floats"
           )
      | A.Bool ->
           (match op with
                  A.Equal ->  L.build_icmp L.Icmp.Eq lval rval ""
builder, m'', builder
                 | A.Neq -> L.build_icmp L.Icmp.Ne lval rval ""
builder, m'', builder
                 | A.And -> L.build_and lval rval "" builder, m'',
builder
                 | A.Or -> L.build_or lval rval "" builder, m'',
builder
                 | _ -> make_err "internal error, cannot perform this
operation on booleans"
           )
      | A.Char ->
           ( match op with
                  A.Equal -> L.build_icmp L.Icmp.Eq lval rval ""
builder, m'', builder
                 | A.Neq -> L.build_icmp L.Icmp.Ne lval rval ""
builder, m'', builder
                 | A.Add    -> binop_char_concat lval rval ""
builder, m'', builder
                 | _ -> make_err "internal error, cannot perform this
operation on characters"
           )
      | A.ArrayList t ->
           ( match op with
                  A.Add    -> let arr, b = binop_array_concat t this
lval rval "" builder in (arr, m'', b)
                 | _ -> make_err "internal error, cannot perform this
operation on characters"
           )
      | A.String ->
           ( match op with
                  A.Equal -> let eq, b = binop_str_equal this lval
rval "" builder in (eq, m'', b)
                 | A.Neq -> let eq, b = binop_str_equal this lval
rval "" builder in
                      (L.build_not eq "" b, m'', b)
```

```
                 | A.Add    -> let n_str, b =  binop_str_concat this
lval rval "" builder in (n_str, m'', b)
                 | _ -> make_err "internal error, cannot perform this
operation on characters"
            )
      | A.Matrix ->
            let lv_p = L.build_alloca matrix_t "" builder in
            let _ = L.build_store lval lv_p builder in
            let width = L.build_extractvalue lval 1 "" builder in
            let height = L.build_extractvalue lval 2 "" builder in
            let size, builder = M.llvm_mat_size width height ""
builder in
            let r_type, _ = rex in
            match r_type with
              Int | Float ->
                 let value = match r_type with Int -> (L.build_sitofp
rval float_t "" builder) | Float -> rval in
                 let output = L.build_alloca matrix_t "" builder in
                 let data_field_loc = L.build_struct_gep output 0 ""
builder in
                 let data_loc = L.build_array_alloca float_t size ""
builder in
                 let width_loc = L.build_struct_gep output 1 ""
builder in
                 let height_loc = L.build_struct_gep output 2 ""
builder in
                 let _ = L.build_store data_loc data_field_loc
builder in
                 let _ = L.build_store width width_loc builder in
                 let _ = L.build_store height height_loc builder in
                 let _ = match op with
                      A.Add -> L.build_call mat_scalar_add_func
[|lv_p; value; output|] "" builder
                      | A.Sub -> L.build_call
mat_scalar_subtract_func [|lv_p; value; output|] "" builder
                      | A.Mult -> L.build_call
mat_scalar_multiply_func [|lv_p; value; output|] "" builder
                      | A.Div -> L.build_call mat_scalar_divide_func
[|lv_p; value; output|] "" builder
                      | A.Exp -> let i_val = L.build_fptosi value
i32_t "" builder in
                           L.build_call mat_mat_power_func [|lv_p;
i_val; output |] "" builder
```

```
                 in
                 let output_v = L.build_load output "" builder in
                 (output_v, m'', builder)
            | Matrix ->
                 let rv_p = L.build_alloca matrix_t "" builder in
                 let _ = L.build_store rval rv_p builder in
                 let r_width = L.build_extractvalue rval 1 "" builder
in
                 let r_height = L.build_extractvalue rval 2 ""
builder in
                 match op with
                   A.Equal -> L.build_call mat_mat_equal_func [|lv_p;
rv_p|] "" builder, m'', builder
                 | _ ->
                      let (o_width, o_height, builder) =
M.llvm_output_size op lval rval builder in
                      let size, builder = M.llvm_mat_size o_width
o_height "" builder in
                      let output = L.build_alloca matrix_t ""
builder in
                      let data_field_loc = L.build_struct_gep output
0 "" builder in
                      let data_loc = L.build_array_malloc float_t
size "" builder in
                      let width_loc = L.build_struct_gep output 1 ""
builder in
                      let height_loc = L.build_struct_gep output 2
"" builder in
                      let _ = L.build_store data_loc data_field_loc
builder in
                      let _ = L.build_store o_width width_loc
builder in
                      let _ = L.build_store o_height height_loc
builder in
                      let _ = match op with
                             A.Add -> L.build_call mat_mat_add_func
[|lv_p; rv_p; output |] "" builder
                           | A.Sub -> L.build_call
mat_mat_subtract_func [|lv_p; rv_p; output |] "" builder
                           | A.Mult -> L.build_call
mat_mat_multiply_func [|lv_p; rv_p; output |] "" builder
                           | A.Div -> L.build_call
mat_mat_divide_func [|lv_p; rv_p; output |] "" builder
```

```
                          | A.Conv -> L.build_call
mat_mat_convolute_func [|lv_p; rv_p; output |] "" builder
                       in
                       let value = L.build_load output "" builder in
                       (value, m'', builder )
        | _ -> make_err "unimplemented"
        )
      | SUnop(_, _) | SAssignAdd(_, _) | SAssignMinus(_, _) |
SAssignTimes(_, _) | SAssignDivide(_, _)
      | SMemberAccess(_, _) -> make_err "Unimplemented"
      | _ -> make_err ("Miss????"^string_of_sexpr (typ,sx))
      in
      let rec stmt map builder (this: L.llvalue) (*Llvm func def*) s =
match s with
        SBlock sl ->
        let b, _ = List.fold_left (fun (b, m) s -> stmt m b this s)
(builder, map) sl in
        (b, map)
      | SExpr e ->
        let (_, m, builder) = (expr map builder this e) in (builder,
m)
      | SDeclare(t, name) ->
        let l_type = ltype_of_typ t in
        let addr = L.build_alloca l_type name builder in
        let m' = StringMap.add name addr map in
        (builder, m')
      | SIf(pred, then_stmt, else_stmt) ->
        let bool_val, m', builder = expr map builder this pred in
        let merge_bb = L.append_block context "merge" this in
        let branch_ins = L.build_br merge_bb in
        let then_bb = L.append_block context "then" this in
        let then_builder, m'' = stmt m' (L.builder_at_end context
then_bb) this then_stmt in
        let () = add_terminal then_builder branch_ins in
        let else_bb = L.append_block context "else" this in
        let else_builder, m'' = stmt m' (L.builder_at_end context
else_bb) this else_stmt in
        let () = add_terminal else_builder branch_ins in
        let _ = L.build_cond_br bool_val then_bb else_bb builder in
        (L.builder_at_end context merge_bb, m')
      | SWhile(predicate, body) ->
        let pred_bb = L.append_block context "while" this in
        let _ = L.build_br pred_bb builder in
```

```
      let body_bb = L.append_block context "while_body" this in
      let body_bldr, m' = stmt map (L.builder_at_end context
body_bb) this body in
      let () = add_terminal body_bldr (L.build_br pred_bb) in
      let pred_bldr = L.builder_at_end context pred_bb in
      let bool_val, m'', pred_bldr = expr m' pred_bldr this
predicate in
      let merge_bb = L.append_block context "merge" this in
      let _ = L.build_cond_br bool_val body_bb merge_bb pred_bldr in
      (L.builder_at_end context merge_bb, m'')
    | SFor(e1, e2, e3, body) -> stmt map builder this ( SBlock [SExpr
e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
    | _ -> make_err "Unimplemented"
    in
    let build_main sl =
      let main_ty = L.function_type i32_t [||] in
      let main_func = L.define_function "main" main_ty code_module
in
      let builder = L.builder_at_end context (L.entry_block
main_func) in
      let builder, _ = stmt StringMap.empty builder main_func
(SBlock sl) in
      ignore(L.build_ret (L.const_int i32_t 0) builder)
    in build_main statements; code_module
```

8.5 sast.ml

Author: Tahmid Munat, Steven Bonilla

```
(* Semantically-checked abstract syntax tree types *)
open Ast

type sexpr = typ * sx
and sx =
     SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
  | SCharLiteral of char
  | SStringLiteral of string
  | SId of string
  | SBinop of sexpr * op * sexpr
```

```
  | SUnop of uop * sexpr
  | SAssign of sexpr * sexpr
  | SAssignAdd of sexpr * sexpr
  | SAssignMinus of sexpr * sexpr
  | SAssignTimes of sexpr * sexpr
  | SAssignDivide of sexpr * sexpr
  | SDeclAssign of typ * string * sexpr
  | SCall of string * sexpr list
  | SArray of sexpr list
  | SArray2D of sexpr list list
  | SArrayIndex of sexpr * sexpr
  | SArray2DIndex of sexpr * sexpr * sexpr
  | SImageIndex of sexpr * string
  | SMemberAccess of sexpr * string list
  | SNoexpr

type sstmt =
     SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SDeclare of typ * string

type sfunc_decl = {
     styp : typ;
     sfname : string;
     sformals : bind list;
     sbody : sstmt list;
  }


type sprogram = sstmt list * sfunc_decl list

(* Pretty-printing functions *)


let rec string_of_sexpr (sex:sexpr) = match snd sex with
     SLiteral(l) -> string_of_int l
  | SFliteral(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
```

```
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> string_of_sexpr v ^ " = " ^ string_of_sexpr e
  | SAssignAdd(v, e) -> string_of_sexpr v ^ " += " ^ string_of_sexpr
e
  | SAssignMinus(v, e) -> string_of_sexpr v ^ " -= " ^
string_of_sexpr e
  | SAssignTimes(v, e) -> string_of_sexpr v ^ " *= " ^
string_of_sexpr e
  | SAssignDivide(v, e) -> string_of_sexpr v ^ " /= " ^
string_of_sexpr e
  | SDeclAssign(t, v, e) -> (string_of_typ t) ^ v ^ " = " ^
string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^
")"
  | SArray(l) -> "[" ^ (String.concat ", " (List.map string_of_sexpr
l)) ^ "]"
  | SArrayIndex(a, b) -> string_of_sexpr a ^ "[" ^ string_of_sexpr b
^ "]"
  | SArray2DIndex(a, b, c) -> string_of_sexpr a ^ "[" ^
string_of_sexpr b ^ "]" ^ "[" ^ string_of_sexpr c ^ "]"
  | SImageIndex(ex, chan) -> string_of_sexpr ex ^ "->" ^ chan
  | SMemberAccess(a, b) -> string_of_sexpr a ^ "." ^String.concat "."
b
  | SCharLiteral(c) -> "'" ^ Char.escaped c ^ "'"
  | SStringLiteral(s) -> "\"" ^ s ^ "\""
  | SNoexpr -> ""


let rec string_of_sstmt = function
      SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^
"}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
```

```
  | SFor(e1, e2, e3, st) ->
      "for " ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ;
"  ^ string_of_sexpr e3 ^" "^ string_of_sstmt st
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
string_of_sstmt s
  | SDeclare(t, s) -> (string_of_typ t) ^ " " ^ s


let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd
fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (stmts, funcs) =
  String.concat "" (List.map string_of_sstmt stmts) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl funcs)
```

8.6 semant.ml

Author: Willie Koomson

```
(* Colode semantic checker *)
open Ast
open Sast

module StringMap = Map.Make(String)
type stmt_context = { current_func: func_decl option }
let check (stmts, functions) =
      let make_err e = raise (Failure e) in
      let add_func map fd =
      let dup_err = "Function with name " ^ fd.fname ^ " is already
defined"
            and name = fd.fname
      in match fd with
            _ when StringMap.mem name map -> make_err dup_err
      | _ -> StringMap.add name fd map
      in
      let built_in_funcs = List.fold_left add_func StringMap.empty [
```

```
      {typ = Void; fname = "print"; formals = [(String, "arg")];
locals = []; body = [] };
      {typ = Void; fname = "iprint"; formals = [(Int, "arg")]; locals
= []; body = [] };
      {typ = Void; fname = "fprint"; formals = [(Float, "arg")];
locals = []; body = [] };
      {typ = Void; fname = "mprint"; formals = [(Matrix, "arg")];
locals = []; body = [] };
      {typ = Matrix; fname = "new"; formals = [(Int, "width"); (Int,
"height")]; locals=[]; body=[]};
      {typ = Image; fname = "coload"; formals = [(String, "arg")];
locals=[]; body=[]};
      {typ = Void; fname = "coclose"; formals = [(Image, "img");
(String, "arg")]; locals=[]; body=[]};
      {typ = Matrix; fname = "generate_gaussian"; formals = [(Int,
"width"); (Int, "height"); (Float, "sigma");]; locals=[]; body=[]};
      {typ = Matrix; fname = "generate_brighten"; formals = [ (Float,
"intensity");]; locals=[]; body=[]};
      {typ = Matrix; fname = "generate_sharpen"; formals = [];
locals=[]; body=[]};
      {typ = Matrix; fname = "generate_edge_detect"; formals = [];
locals=[]; body=[]};
      ]  (* TODO add other standard library functions*)
      in
      let func_decls = List.fold_left add_func built_in_funcs
functions  in
      let find_func name =
      try StringMap.find name func_decls
      with Not_found -> raise( Failure("Undeclared function: " ^
name))
      in
      let add_var map ventry =
      let name = snd ventry in
      let dup_err = "Variable with name " ^ name ^" is a duplicate."
in
      match ventry with
            _ when StringMap.mem name map -> make_err dup_err
      | _ -> StringMap.add name ventry map
      in
      let find_var map name =
      try StringMap.find name map
      with Not_found -> raise( Failure("Undeclared variable: " ^
name))
```

```
in
(* UNUSED let check_var_decl var map =
let void_err = "Illegal void " ^ snd var
     and dup_err = "Duplicate declaration: " ^ snd var
in match var with
     (Void, _) -> raise (Failure void_err)
| (typ, id) -> match (StringMap.find_opt id map) with
     Some v -> raise (Failure dup_err)
     | None -> SDeclare(typ, id)
in *)
let check_type_equal lvaluet rvaluet err =
if lvaluet = rvaluet then lvaluet else raise (Failure err)
in
let type_of_id map id = fst (find_var map id) in
let rec check_expr map exp  = match exp with
Literal l -> (Int, SLiteral l, map)
| Fliteral l -> (Float, SFliteral l, map)
| BoolLit l -> (Bool, SBoolLit l, map)
| CharLiteral  l -> (Char, SCharLiteral l, map)
| StringLiteral s -> (String, SStringLiteral s, map)
| Id i -> (type_of_id map i, SId i, map)
| Unop(op, e) as ex ->
let (t, sx, map') = check_expr map e  in
let ty = match op with
     Neg when t = Int || t = Float || t = Image || t = Matrix
-> t
     | Not when t = Bool || t = Matrix -> t
     | _ -> make_err ("Illegal unary operator" ^ string_of_uop
op ^ string_of_typ t ^ "in" ^ string_of_expr ex)
in (ty, SUnop(op, (t, sx)), map')
| Binop(e1, op, e2) as ex ->
let (t1, e1', map') = check_expr map e1
in let (t2, e2', map'') = check_expr map' e2
in
let same = t1 = t2 in
let matrix_scalar = (t2 = Int || t2 = Float) in
let ty =
match t1 with
| ArrayList inner -> (match op with
          Add -> t1
          | _ -> make_err ("Illegal binary operation, cannot
perform "^string_of_expr ex^" on lists."))
| _ -> match op with
```

```
          Add | Sub | Mult | Div | Exp when same && t1 = Int   ->
Int
          | Add | Sub | Mult | Div | Exp when same && t1 = Float ->
Float
          | Add when same && t1 = Char -> Char
          | Add when same && t1 = String -> String
          | Add | Sub | Mult | Div | Conv when same && t1 = Matrix
-> Matrix
          | Add | Sub | Mult | Div when t1 = Matrix && matrix_scalar
-> Matrix
          | Exp when t1 = Matrix && t2 = Int -> Matrix
          | Equal | Neq        when same            -> Bool
          | Less | Leq | Greater | Geq
                    when same && (t1 = Int || t1 = Float) -> Bool
          | And | Or when same && t1 = Bool -> Bool
          | _ -> make_err ("Illegal binary operator " ^
string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^ string_of_typ t2 ^ "
in " ^ string_of_expr ex)
     in (ty, SBinop((t1, e1'), op, (t2, e2')), map'')
     | Assign(name, e) as ex ->
     let err = "illegal assignment " ^ string_of_expr ex in
     let (left_t, sname, map') = check_name name map err in
     let (right_t, sx, map'') = check_expr map' e in
     (check_type_equal left_t right_t err, SAssign((left_t, sname),
(right_t, sx)), map'')
     | AssignAdd (name, e) as ex ->
     let err = "illegal assign-add " ^ string_of_expr ex in
     let (left_t, sname, map') = check_name name map err in
     let (right_t, sx, map'') = check_expr map' e in
     let ty = match left_t with
          Matrix -> (match right_t with Int | Float -> Float |
Matrix -> Matrix | _ -> make_err err)
          | _ -> check_type_equal left_t right_t err
     in (match ty with
               Int | Float | Matrix | String -> (ty,
SAssign((left_t, sname), (left_t, SBinop((left_t, sname), Add,
(right_t, sx)))), map'')
               | _ -> make_err err)
     | AssignMinus (name, e) as ex ->
     let err = "illegal assign-minus " ^ string_of_expr ex in
     let (left_t, sname, map') = check_name name map err in
     let (right_t, sx, map'') = check_expr map' e in
     let ty = match left_t with
```

```
          Matrix -> (match right_t with Int | Float -> Float |
Matrix -> Matrix | _ -> make_err err)
          | _ -> check_type_equal left_t right_t err
     in (match ty with
               Int | Float | Matrix -> (ty, SAssignMinus((left_t,
sname), (right_t, sx)), map'')
               | _ -> make_err err)
     | AssignTimes (name, e) as ex ->
     let err = "illegal assign-times " ^ string_of_expr ex in
     let (left_t, sname, map') = check_name name map err in
     let (right_t, sx, map'') = check_expr map' e in
     let ty = match left_t with
          Matrix -> (match right_t with Int | Float -> Float |
Matrix -> Matrix | _ -> make_err err)
          | _ -> check_type_equal left_t right_t err
     in (match ty with
               Int | Float | Matrix -> (ty, SAssignTimes((left_t,
sname), (right_t, sx)), map'')
               | _ -> make_err err)
     | AssignDivide (name, e) as ex ->
     let err = "illegal assign-divide " ^ string_of_expr ex in
     let (left_t, sname, map') = check_name name map err in
     let (right_t, sx, map'') = check_expr map' e in
     let ty = match left_t with
          Matrix -> (match right_t with Int | Float -> Float |
Matrix -> Matrix | _ -> make_err err)
          | _ -> check_type_equal left_t right_t err
     in (match ty with
               Int | Float | Matrix -> (ty, SAssignDivide((left_t,
sname), (right_t, sx)), map'')
               | _ -> make_err err)
     | DeclAssign (left_t, id, e) as ex ->
     let (right_t, sx, map') = check_expr map e  in
     let err = "illegal argument found. LHS is " ^ string_of_typ
left_t ^ ", while RHS is "^string_of_typ right_t^", types must match
in assignment for "^string_of_expr ex  in
     let ty = check_type_equal left_t right_t err in
     let new_map = add_var map' (ty, id) in
     let right = (right_t, sx) in
     let da = SDeclAssign(ty, id, right) in
     (ty, da, new_map)
     | Call(func, args) as call ->
     let fd = find_func func in
```

```
      let param_length = List.length fd.formals in
      if List.length args != param_length then
      make_err ("expecting " ^ string_of_int param_length ^ "
arguments in " ^ string_of_expr call)
      else let check_call (param_t, _) e =
            let (arg_t, sx, _) = check_expr map e in
            let err = "illegal argument found " ^ string_of_typ arg_t
^
            " expected " ^ string_of_typ param_t ^ " in " ^
string_of_expr e
            in (check_type_equal param_t arg_t err, sx)
      in
      let args' = List.map2 check_call fd.formals args
      in (fd.typ, SCall(func, args'), map)
      | Array(l) as exp ->
      if List.length l = 0 then (Void, SArray([]), map) (* make sure
assignment allows an empty array*)
      else let sbody = List.map (check_expr map) l in
            let err = "Illegal array literal, arrays are single type
in " ^ string_of_expr exp in
            let match_type, _, _ = List.nth sbody 0 in
            let correct = List.for_all (fun (t, _, _) -> t =
match_type) sbody in
            if correct then
                  let clean_body = List.map (fun (t, sx, _) -> (t,sx))
sbody in
                  (ArrayList match_type, SArray(clean_body), map)
            else make_err err
      | Array2D(l) as exp ->
      let row_lens = List.map List.length l in
      let length = List.hd row_lens in
      let equal = List.for_all (fun a -> a = length) row_lens in
      if not equal then
            let err =  (string_of_expr exp) ^": matrix row lengths
must be equal" in
            make_err err
      else let check_row r =
                  let row_body = List.map (check_expr map) r in
                  List.map (fun (t, sx, _) -> (t,sx)) row_body
            in
            let rows = List.map check_row l in
      (Matrix, SArray2D(rows), map)
      | ArrayIndex(name, idx) ->
```

```
      let cannot_idx_err = "Illegal index on " ^ string_of_expr name
in
      let invalid_idx_err = "Illegal index on " ^ string_of_expr name
^ ". Index must be numerical" in
      let (typ, sid, map') = match name with
            Id _ -> check_expr map name
            | _ -> make_err cannot_idx_err
      in
      let inner_typ = match typ with
            ArrayList lt -> lt
            | Pixel -> Float (* TODO support 1d index on matrix, inner
type is then List *)
            | _ -> make_err cannot_idx_err
      in
      let (idx_type, si, map'') = match idx with
            Literal _ -> check_expr map' idx
            | _ -> make_err invalid_idx_err
      in
      let arr = (typ, sid) in
      let index = (idx_type, si) in
      (inner_typ, SArrayIndex(arr, index), map'')
      | Array2DIndex (name, idx, idx2) ->
      let cannot_idx_err = "Illegal index on " ^ string_of_expr name
^ " in " in
      let invalid_idx_err = "Illegal index on " ^ string_of_expr name
^ ". Index must be numerical" in
      let (typ, sid, map') = match name with
            Id _ -> check_expr map name
            | _ -> make_err cannot_idx_err
      in
      let inner_typ = match typ with
            Matrix -> Float
            | _ -> make_err cannot_idx_err
      in
      let (idx_type, si, map'') = match idx with
            Id _ |  Literal _ -> check_expr map' idx
            | _ -> make_err invalid_idx_err
      in
      let (idx2_type, si2, map''') = match idx2 with
            Id _ |  Literal _ -> check_expr map'' idx2
            | _ -> make_err invalid_idx_err
      in
      if (idx_type != Int) || (idx2_type != Int) then
```

```
          make_err invalid_idx_err
     else
          let mat = (typ, sid) in
          let index = (idx_type, si) in
          let index2 = (idx2_type, si2) in
          (inner_typ, SArray2DIndex(mat, index, index2), map''')
     | ImageIndex(id, channel) ->
     let invalid_chan_err = channel ^ " is not a valid channel on "
^ (string_of_expr id) ^ ". Use red, green, blue, or alpha." in
     let (typ, sid, map') = match id with
          Id _ -> check_expr map id
          | _ -> make_err "Cannot get the member of non-image
variable."
     in
     let valid_channels = ["red"; "green"; "blue"; "alpha"] in
     if not (List.mem channel valid_channels) then make_err
invalid_chan_err
     else (Matrix, SImageIndex((typ, sid), channel), map')
     | MemberAccess(_, _) ->  (Void, SNoexpr, map) (* Todo *)
     | Noexpr -> (Void, SNoexpr, map)
     and check_name (name : expr) map err : (Ast.typ * Sast.sx *
(Ast.typ * StringMap.key) StringMap.t
) = match name with
     Id _ | ArrayIndex(_,_) | Array2DIndex(_,_,_) | ImageIndex(_,_)
-> check_expr map name
     | _ -> make_err err
     in
     let check_bool_expr map e =
     let (t', e', map') = check_expr map e
     and err = "expected Boolean expression in " ^ string_of_expr e
     in if t' != Bool then raise (Failure err) else (t', e')
     in
     let rec check_stmt map st (ctxt : stmt_context) = match st with
     Expr e -> let (ty, sx, map') = check_expr map e in (SExpr (ty,
sx), map')
     | Return e -> let (ty, sx, map') = check_expr map e in
     let return_from_global_err = "Cannot return " ^ string_of_expr
e ^ " from global context" in
     (match ctxt.current_func with
               None -> if ty <> Void then make_err
return_from_global_err else (SReturn(ty, sx), map')
```

```
                | Some(fd) -> (* UNUSED let invalid_return_err =
"return gives " ^ string_of_typ ty ^ " expected " ^ string_of_typ
fd.typ ^ " in " ^ string_of_expr e in  *)
                    if ty = fd.typ then (SReturn((ty, sx)), map')
                    else make_err return_from_global_err)
        | If(pred, then_block, else_block) ->
        let sthen, _ = check_stmt map then_block ctxt in
        let selse, _ = check_stmt map else_block ctxt in
        (SIf(check_bool_expr map pred, sthen, selse), map)
        | For(e1, e2, e3, st) ->
        (* let invalid_err = "Invalid for loop cursor" in
        let invalid_iterator_err = "Invalid for loop iterator" in
        let err = "Name of for loop cursor already in use:" ^
string_of_expr cursor in
        let check_iterator map iterator =
            let (ty, sx, map') = check_expr map iterator in
            match ty with
            ArrayList _ | Pixel | Matrix | String -> (ty, sx, map')
            | _ -> make_err invalid_iterator_err
        in
        let (ty, sx, map') = check_iterator map iterator in
        let name = match cursor with
            Id n -> n | _ -> make_err invalid_err
        in
        if StringMap.mem name map' then make_err err else
        let it_ty = match ty with
            ArrayList(t) -> t | Pixel | Matrix -> Float | String ->
Char | _ -> make_err invalid_iterator_err
        in
        let new_map = add_var map' (it_ty, name) in
        let (sblock, _) = check_stmt new_map block ctxt in *)
        let (ty1, sx1, m') = check_expr map e1 in
        let (ty3, sx3, m'') = check_expr m' e3 in
        SFor((ty1,sx1), check_bool_expr map e2, (ty3, sx3), fst
(check_stmt map st ctxt)), map
        | While(p, s) -> SWhile(check_bool_expr map p, fst (check_stmt
map s ctxt)), map
        | Declare(t, id) ->
        let new_map = add_var map (t, id) in
        (SDeclare(t, id), new_map)
        | Block stl ->
        let (checked, map') = check_stmt_list map stl ctxt in
        (SBlock(checked), map)
```

```
    and check_stmt_list map sl (ctxt : stmt_context) = match sl
with
    [Return _ as s] -> ([fst (check_stmt map s ctxt)], map)
    | Return _ :: _   -> raise (Failure "nothing may follow a
return")
    | Block sl :: ss  -> check_stmt_list map (sl @ ss) ctxt(*
Flatten blocks *)
    | s :: ss        -> let (sst, map') = check_stmt map s ctxt in
         let (slist, map'') = check_stmt_list map' ss ctxt in
         (sst :: slist, map'')
    | []             -> ([], map)
    in
    let check_func fd =
    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add
name (ty, name) m) StringMap.empty fd.formals
    in
    {
         styp = fd.typ;
         sfname = fd.fname;
         sformals = fd.formals;
         sbody = let (blk, map') = check_stmt symbols
(Block(fd.body)) {current_func= Some fd} in
              match blk with SBlock(s1) -> s1
              | _ -> make_err "Internal err... block didn't become
block?";
    }
    in
    let sfunctions = List.map check_func functions in
    let (sstmt, _) = check_stmt_list StringMap.empty stmts
{current_func= None} in
    (sstmt, sfunctions)
```

8.7 toplevel.ml

Author: Takuma Yasuda

```
(* Top-level of the compiler: scan & parse the input, check the
resulting AST, generate LLVM IR, and dump the mudule *)
type action = Ast | Sast | LLVM_IR
let () =
    let action = ref LLVM_IR in
    let set_action a () = action := a in
```

```
    let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM
IR");
    ] in
    let usage_msg = "usage: ./colode.native [-a|-s|-l] [file.cld]" in
    let channel = ref stdin in
    Arg.parse speclist (fun file -> channel := open_in file)
usage_msg;
    let lexbuf = Lexing.from_channel !channel in
    let ast = Parser.program Scanner.token lexbuf in
    match !action with
      Ast -> print_string (Ast.string_of_program ast)
    | _ -> let sast = Semant.check ast in
      match !action with
        Ast -> ()
      | Sast -> print_string (Sast.string_of_sprogram sast)
      | LLVM_IR -> print_string (Llvm.string_of_llmodule
(Codegen.translate sast))

(*
type action = Ast | LLVM_IR | Compile

let _ =
      let action = if Array.length Sys.argv > 1 then
                List.assoc Sys.argv.(1) [ ("-a", Ast);
                          ("-l", LLVM_IR);
                          ("-c", Compile)]
      else Compile in
      let lexbuf = Lexing.from_channel stdin in
      let ast = Parser.program Scanner.token lexbuf in
      (match action with
                Ast -> print_string (Ast.string_of_program ast)
                | LLVM_IR -> print_string ("LLVM_IR not implemented
yet")
                | Compile -> print_string ("Compile not implemented
yet"))
*)
```

8.5 matrix.h

Author: Dimitri Borgers and Willie Koomson

```
struct mat {
    double* d;
    int width;
    int height;
};

int _mat_index(int width, int i, int j);
void _mat_reset(struct mat* m);
void _mat_zero_out(struct mat* m);
void _mat_print(const struct mat* m);
void _mat_scalar_add(struct mat* m, double a, struct mat* out);
void _mat_scalar_subtract(struct mat* m, double a, struct mat* out);
void _mat_scalar_multiply(struct mat* m, double a, struct mat* out);
void _mat_scalar_divide(struct mat* m, double a, struct mat* out);
void _mat_mat_add(struct mat* l, struct mat* r, struct mat* out);
void _mat_mat_subtract(struct mat* l, struct mat* r, struct mat*
out);
void _mat_mat_multiply(struct mat* l, struct mat* r, struct mat*
out);
void _mat_make_identity(struct mat* m);
void _mat_mat_inverse(struct mat* l, struct mat* out);
void _mat_mat_power(struct mat* l, int a, struct mat* out);
void _mat_mat_convolute(struct mat* a, struct mat* b, struct mat*
out);
int _mat_mat_equal(struct mat* l, struct mat* r);
void _mat_gen_gauss(double sigma, struct mat* out);
```

8.6 matrix.c

Authors: Willie Koomson

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "matrix.h"


int _mat_index(int width, int i, int j) {
    return i*width + j;
}

void _mat_reset(struct mat* m) {
```

```
    m->width = 0;
    m->height = 0;
}

void _mat_zero_out(struct mat* m) {
    int size = m->height*m->width;
    for (int i = 0; i < size; ++i)
    {
      m->d[i] = 0.0;
    }
}

void _mat_print(const struct mat* m) {
    printf("%d x %d [", m->height, m->width);
    for (int i = 0; i < m->height; ++i)
    {
      for (int j = 0; j < m->width; ++j)
      {
           int idx = _mat_index(m->width,i,j);
           printf("%03.2f ", m->d[idx]);
      }
      if (i != (m->height-1)) printf("|\n");
    }
    printf(" ]\n");
}

void _mat_scalar_add(struct mat* m, double a, struct mat* out) {
    for (int i = 0; i < m->height; ++i)
    {
      for (int j = 0; j < m->width; ++j)
      {
           int idx = _mat_index(m->width,i,j);
           out->d[idx] = m->d[idx] + a;
      }
    }
}

void _mat_scalar_subtract(struct mat* m, double a, struct mat* out) {
    for (int i = 0; i < m->height; ++i)
    {
      for (int j = 0; j < m->width; ++j)
      {
           int idx = _mat_index(m->width,i,j);
```

```
            out->d[idx] = m->d[idx] - a;
        }
    }
}

void _mat_scalar_multiply(struct mat* m, double a, struct mat* out) {
    for (int i = 0; i < m->height; ++i)
    {
      for (int j = 0; j < m->width; ++j)
      {
            int idx = _mat_index(m->width,i,j);
            out->d[idx] = m->d[idx] * a;
      }
    }
}

void _mat_scalar_divide(struct mat* m, double a, struct mat* out) {
    for (int i = 0; i < m->height; ++i)
    {
      for (int j = 0; j < m->width; ++j)
      {
            int idx = _mat_index(m->width,i,j);
            out->d[idx] = m->d[idx] / a;
      }
    }
}

void _mat_mat_add(struct mat* l, struct mat* r, struct mat* out) {
    if ((l->width != r->width) || (l->height != r->height)) {
      printf("ERROR: Illegal attempt to add %dx%d matrix and %dx%d
matrix.", l->height, l->width, r->height, r->width);
      _mat_reset(out);
    }
    else {
      for (int i = 0; i < l->height; ++i)
      {
            for (int j = 0; j < l->width; ++j)
            {
                  int idx = _mat_index(l->width,i,j);
                  out->d[idx] = l->d[idx] + r->d[idx];
            }
      }
    }
```

```
}
void _mat_mat_subtract(struct mat* l, struct mat* r, struct mat* out)
{
    if ((l->width != r->width) || (l->height != r->height)) {
      printf("ERROR: Illegal attempt to add %dx%d matrix and %dx%d
matrix. Operation skipped and output ", l->height, l->width,
r->height, r->width);
      _mat_reset(out);
    }
    else {
      for (int i = 0; i < l->height; ++i)
      {
           for (int j = 0; j < l->width; ++j)
           {
                int idx = _mat_index(l->width,i,j);
                out->d[idx] = l->d[idx] - r->d[idx];
           }
      }
    }
}
void _mat_mat_multiply(struct mat* l, struct mat* r, struct mat* out)
{
    if (l->width != r->height) {
      printf("ERROR: Illegal attempt to multiply %dx%d matrix and
%dx%d matrix. The second operand should have been %dx%d",
           l->height, l->width, r->height, r->width, l->width,
r->width);
      _mat_reset(out);
      return;
    }
    int size = l->height*r->width;
    double d[size];
    struct mat copy = {d, r->width, l->height};
    for (int i = 0; i < l->height; ++i)
    {
      for (int j = 0; j < r->width; ++j)
      {
           double sum = 0;
           for (int k = 0; k < r->height; ++k) {
                int l_idx = _mat_index(l->width,i,k);
                int r_idx = _mat_index(r->width,k,j);
                sum +=  l->d[l_idx] * r->d[r_idx];
           }
```

```
            int idx = _mat_index(copy.width, i, j);
            copy.d[idx] = sum;
        }
    }
    for (int i = 0; i < size; ++i) {
        out->d[i] = copy.d[i];
    }
}

int _mat_mat_equal(struct mat* l, struct mat* r) {
    if ((l->width != r->width) || (l->height != r->height)) {
        return 0;
    }
    int equal = 1;
    for (int i = 0; i < l->height; ++i)
    {
        for (int j = 0; j < r->width; ++j)
        {
            int idx = _mat_index(l->width, i, j);
            if (l->d[idx] != r->d[idx]) {
                equal = 0;
            }
        }
    }
    return equal;
}
void _mat_make_identity(struct mat* m) {
    int count = 0;
    for (int i = 0; i < m->height; ++i)
    {
        for (int j = 0; j < m->width; ++j)
        {
            int idx = _mat_index(m->width, i, j);
            if (j == i) {
                m->d[idx] = 1.0;
                count++;
            } else {
                m->d[idx] = 0.0;
            }
        }
    }
}
```

```
void _mat_mat_inverse(struct mat* l, struct mat* out) {
    double rcond = 1E-15; // numpy default
}
void _mat_mat_power(struct mat* l, int a, struct mat* out) {
    if (l->width != l->height) {
      printf("ERROR: cannot take the power of a non-square
matrix.\n");
      return;
    }
    int i_len = l->width;
    double d[i_len * i_len];
    struct mat ident = {d, i_len, i_len};
    _mat_make_identity(&ident);
    if (a == 0) {
      out->width = ident.width;
      out->height = ident.height;
      memcpy(out->d, ident.d, i_len * i_len);
      return;
    }
    _mat_mat_multiply(l, &ident, out);
    for (int i = 1; i < a; ++i)
    {
      _mat_mat_multiply(l, out, out);
    }
    if (a < 0) {
      _mat_mat_inverse(out, out);
    }
}

int convolve2D(struct mat* a, struct mat* kernel, struct mat* out) {
    int i, j, m, n;
     double *in_p, *in_p2, *out_p, *out_p2, *kern_p;
     int kern_center_x, kern_center_y;
     int row_min, row_max;                           // to check
boundary of input array
     int col_min, col_max;
     int a_x = a->width;
     int a_y = a->height;
     int kern_x = kernel->width;
     int kern_y = kernel->height;
     kern_center_x = kern_x >> 1; // half of kern width
     kern_center_y = kern_y >> 1; // half of kern width
```

```
	in_p = in_p2 = a->d + _mat_index(a_x, kern_center_y,
kern_center_x);
	out_p = out->d;
	kern_p = kernel->d;

	for (int i = 0; i < a_y; ++i){
	 row_max = i + kern_center_y;
	 row_min = i - a_y + kern_center_y;
	 for (int j = 0; j < a_x; ++j) {
		col_max = j + kern_center_x;
		col_min = j - a_x + kern_center_x;
		*out_p = 0;
		// flip kernel and multiply
		for (int l = 0; l < kern_y; ++l) {
			if (l <= row_max && l > row_min) {
				for (m = 0; m < kern_x; ++m) {
					if (m <= col_max && m > col_min) {
						*out_p += *(in_p - m) * (*kern_p);
					}
					kern_p++;
				}
			} else {
				kern_p += kern_x;
			}
			in_p -= a_x;
		}
		if (*out_p < 0) *out_p = 0;
		kern_p = kernel->d;
		++in_p2;
		in_p = in_p2;
		++out_p;
	 }
	}
	// for (int i = 0; i < a_x * a_y ; ++i)
	//   {
	//     out->d[i] = 0xe;
	//   }
	//   _mat_print(out);
	return 1;
}
void _mat_mat_convolute(struct mat* a, struct mat* b, struct mat*
out) {
    convolve2D(a, b, out);
```

```
}

void _mat_gen_gauss(double sigma, struct mat* out) {
    double r, s = 2.0 * sigma * sigma;
    double sum = 0.0;
      double half_i = floor(out->width / 2);
      double half_j = floor(out->height / 2);
     // generating 5x5 kernel
     for (int i = 0; i < out->width; i++)
     {
     for(int j = 0; j < out->height; j++)
     {
          r = sqrt(pow(i-half_i, 2) + pow(j-half_j, 2));
          int idx = _mat_index(out->width, i, j);
          out->d[idx] = (exp(-(r*r)/s))/(M_PI * s);
          sum += out->d[idx];
     }
     }

     // normalising the Kernel
     for (int i = 0; i < out->width; ++i)
     for (int j = 0; j < out->height; ++j) {
           out->d[_mat_index(out->width, i, j)] /= sum;
     }
}

void _mat_gen_sharpen(struct mat* out) {
     int mid_idx = _mat_index(out->width, out->width / 2,
out->height / 2);
     out->d[mid_idx] = 5;
     out->d[_mat_index(out->width, out->width / 2, 0)] = -1.0;
     out->d[_mat_index(out->width, 0, out->height / 2)] = -1.0;
     out->d[_mat_index(out->width, out->width - 1, out->height / 2)]
= -1.0;
     out->d[_mat_index(out->width, out->width / 2, out->height -1)]
= -1.0;
}
void _mat_gen_brighten(double value, struct mat* out) {
    int mid_idx = _mat_index(out->width, out->width / 2, out->height
/ 2);
    out->d[mid_idx] = value;
}
void _mat_gen_edge_detect(struct mat* out) {
```

```
    for (int i = 0; i < out->width; i++)
     {
     for(int j = 0; j < out->height; j++)
     {
          int idx = _mat_index(out->width, i, j);
          out->d[idx] = -1.0;
     }
     }
     int mid_idx = _mat_index(out->width, out->width / 2,
out->height / 2);
     out->d[mid_idx] = 8.0;
}
```

8.7 image.c

Author: Willie Koomson

```
#include "matrix.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define PNG_DEBUG 3
#include <png.h>

struct image {
    int width;
    int height;
    struct mat red;
    struct mat green;
    struct mat blue;
    struct mat alpha;
};

void _image_read(const char* filename, struct image* out) {
    char header[8];
    png_structp png_p;
    png_infop info_p;
    FILE *fp = fopen(filename, "rb");
    if (!fp) {
```

```
        printf("ERROR: could not read %s. Make sure the file
exists\n", filename);
        abort();
     }
     fread(header, 1, 8, fp);
      if (png_sig_cmp(header, 0, 8)) {
        printf("ERROR: could not read %s. Make sure the file is a
valid .png file.\n", filename);
        abort();
      }
      png_p = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
      if (!png_p) {
        printf("ERROR: could not read %s. Make sure the file is a
valid .png file.\n", filename);
        abort();
      }
      info_p = png_create_info_struct(png_p);
      if (!info_p) {
        printf("ERROR: could not read info_strct from %s. Make sure
the file is a valid .png file.\n", filename);
        abort();
      }
      if (setjmp(png_jmpbuf(png_p))) {
        printf("ERROR: could not read info_strct from %s. Make sure
the file is a valid .png file.\n", filename);
        abort();
      }
      png_init_io(png_p, fp);
      png_set_sig_bytes(png_p, 8);
      png_read_info(png_p, info_p);
      int width = png_get_image_width(png_p, info_p);
      int height = png_get_image_height(png_p, info_p);
      png_byte color_type = png_get_color_type(png_p, info_p);
      png_byte bit_depth = png_get_bit_depth(png_p, info_p);
      int number_of_passes = png_set_interlace_handling(png_p);
      // png_read_update_info(png_p, info_p);
      /* read file */
     if (setjmp(png_jmpbuf(png_p))) {
        printf("ERROR: idk really...");
        abort();
     }
```

```
    if(bit_depth == 16)
      png_set_strip_16(png_p);

    if(color_type == PNG_COLOR_TYPE_PALETTE)
      png_set_palette_to_rgb(png_p);

    // PNG_COLOR_TYPE_GRAY_ALPHA is always 8 or 16bit depth.
    if(color_type == PNG_COLOR_TYPE_GRAY && bit_depth < 8)
      png_set_expand_gray_1_2_4_to_8(png_p);

    if(png_get_valid(png_p, info_p, PNG_INFO_tRNS))
      png_set_tRNS_to_alpha(png_p);

    // These color_type don't have an alpha channel then fill it with
0xff.
    if(color_type == PNG_COLOR_TYPE_RGB ||
     color_type == PNG_COLOR_TYPE_GRAY ||
     color_type == PNG_COLOR_TYPE_PALETTE)
      png_set_filler(png_p, 0xFF, PNG_FILLER_AFTER);

    if(color_type == PNG_COLOR_TYPE_GRAY ||
     color_type == PNG_COLOR_TYPE_GRAY_ALPHA)
      png_set_gray_to_rgb(png_p);

    png_read_update_info(png_p, info_p);

    png_bytep* row_pointers = (png_bytep*) malloc(sizeof(png_bytep) *
height);
    for (int i=0; i<height; i++)
           row_pointers[i] = (png_byte*)
malloc(png_get_rowbytes(png_p,info_p));

    png_read_image(png_p, row_pointers);
    fclose(fp);
    int size = sizeof(double)*height*width;
    int malloc_count = 0;
    double* channel_d[4];
    while (malloc_count != 4) {
      channel_d[malloc_count] = malloc(size);
      if (channel_d[malloc_count] != NULL) { malloc_count++; }
    }
    for (int i = 0; i < height; ++i)
    {
```

```
        png_byte* row = row_pointers[i];
        for (int j = 0; j < width; ++j)
        {
             png_byte* ptr = row + (j*4);
             int idx = _mat_index(width, i, j);
             channel_d[0][idx] = ptr[0];
             channel_d[1][idx] = ptr[1];
             channel_d[2][idx] = ptr[2];
             channel_d[3][idx] = ptr[3];
        }
        free(row);
    }
    free(row_pointers);
    out->width = width;
    out->height = height;
    out->red = (struct mat) {channel_d[0], width, height };
    out->green = (struct mat) {channel_d[1], width, height };
    out->blue = (struct mat) {channel_d[2], width, height };
    out->alpha = (struct mat) {channel_d[3], width, height };
    // _mat_print(&out->channels[1]);
}

void _image_write(const char* filename, struct image* in) {
    int height = in->height;
    int width = in->width;
    FILE *fp = fopen(filename, "wb");
    if(!fp) {
        printf("ERROR: coclose() could not write %s", filename);
        abort();
    }
    png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
    if (!png) abort();

    png_infop info = png_create_info_struct(png);
    if (!info) abort();

    if (setjmp(png_jmpbuf(png))) abort();

    png_init_io(png, fp);

    // Output is 16bit depth, RGBA format.
    png_set_IHDR(
```

```
        png,
        info,
        width, height,
        16,
        PNG_COLOR_TYPE_RGBA,
        PNG_INTERLACE_NONE,
        PNG_COMPRESSION_TYPE_DEFAULT,
        PNG_FILTER_TYPE_DEFAULT
    );

    png_write_info(png, info);
    png_bytep* row_pointers = (png_bytep*) malloc(sizeof(png_bytep) *
height);
    for (int i=0; i < height; i++)
            row_pointers[i] = (png_byte*)
malloc(png_get_rowbytes(png,info));
    // copy from image matrices
    for (int i = 0; i < height; ++i)
    {
      png_byte* row = row_pointers[i];
      for (int j = 0; j < width; ++j)
      {
            png_byte* ptr = row + (j*8);
            int idx = _mat_index(width, i, j);
            ((unsigned short*)ptr)[0] = (unsigned short)
in->red.d[idx];
            ((unsigned short*)ptr)[1] = (unsigned short)
in->green.d[idx];
            ((unsigned short*)ptr)[2] = (unsigned short)
in->blue.d[idx];
            ((unsigned short*)ptr)[3] = (unsigned short)
in->alpha.d[idx];
            for (int c = 0; c < 4; c++) {
            }
            /*ptr[0] = in->channels[0].d[idx];
            ptr[1] = in->channels[1].d[idx];
            ptr[2] = in->channels[2].d[idx];
            ptr[3] = in->channels[3].d[idx];*/
      }
    }
    png_write_image(png, row_pointers);
    png_write_end(png, NULL);
```

```
    for (int i = 0; i < height; i++) {
        free(row_pointers[i]);
    }
    free(row_pointers);
    fclose(fp);
}
```