pixelman Final Report

Anthony Chan, Teresa Choe, Gabriel Kramer-Garcia, Brian Tsau

December 2017

Contents

1 Introduction

pixelman is a language used mainly for matrix manipulation. At its core are vectors and matrices, making it much more intuitive and easier to compute vector/matrix operations. By connecting it with both a built-in library and a C library, it is also used to manipulate raster graphics by changing individual pixels or bitmaps.

In other languages, editing matrices/array values can be a tedious task. Even more so, editing images can be very complex and require bulkier software. **pixelman** aims to create solutions for these problems by making a more intuitive interface that allows users to write few lines of codes in very little time.

2 Language Tutorial

pixelman uses a syntax that is similar to Java or C with a few exceptions.

2.1 Statements and Variable declaration

Every statement must end with a semi-colon. A newly declared variable is preceded by its type. All variable declarations in a function body must be before its expressions.

```
int a;
a = 5;
print_int(5);
```

2.2 Functions

Functions are declared with the def keyword followed by the return type, followed by the function name, followed by any parameters in a set of parentheses. The contents of the function must be put inside of a set of curly brackets.

```
def int multiplyByFive(int factor){
    int product;
    product = factor * 5;
    return product;
}
```

2.3 Main method

Every executable program must have a main() method. The main() method is the first function that gets called by the CPU, and determines the rest of the program execution.

```
def void main(){
    print_string("Hello World!");
}
```

2.4 Vectors

Vectors are similar to arrays in Java. Vectors can be declared, assigned a value, and accessed as seen below.

```
def int main(){
    int[5] myVector;
    myVector = [1,2,3,4,5];
    print_int(myVector[2]); :)prints 3
}
```

2.5 Matrices

Matrices are similar to 2-dimensional arrays in Java. A matrix can be declared, assigned a value, and accessed as seen below. A matrix literal has enclosing brackets denoted by [||]. Inside the literal are arrays delimited by ampersand.

Our languages only work for 3 by 3 matrices. The reason is because most image convolution is done with 3 by 3 matrices.

```
def int main(){
    int[3][3] myMatrix;
    myMatrix = [|[1,2,3]&[4,5,6]&[11,12,13]|}
    print_int(myMatrix[2][1]); :)prints 12
}
```

3 Language Reference Manual

3.1 Introduction

pixelman is a language used mainly for matrix manipulation. At its core are vectors and matrices, making it much more intuitive and easier to compute vector/matrix operations. By connecting it with both a built-in library and a C library, it is also used to manipulate raster graphics by changing individual pixels or bitmaps.

3.2 Lexical Conventions

There are six kinds of tokens: identifiers, keywords, constants, expression operators, and other separators.

3.2.1 Comments

The character sequence `:)` introduces a single line comment. Multi-line comments begin with the character sequence `(:` and end with the character sequence `:)`.

3.2.2 Identifiers

An identifier is a sequence of letters and digits that labels variables, functions, and classes; the first character must be alphabetic. The underscore _ and dash - are accepted in an identifier. **pixelman** is case sensitive so upper and lower case letters are considered different.

3.2.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

`void`	`string`	`true`
`return`	`bool`	`false`
`def`	`if`	
`main`	`else`	`Vector`
`int`	`for`	
`float`	`while`	`Matrix`

3.2.4 Literals

Literals are notations for constant values of some built-in types.

3.2.4.1 Integer literals An integer literal is a sequence of digits, e.g `2`, `42`, `108`, or `-14`.

3.2.4.2 Floating point literals A floating literal consists of an integer part, a required decimal point, and a fraction part. There can be no integer part or no fraction part, but at least one is required. The integer and fraction parts both consist of a sequence of digits, e.g. `1.0`, `.1`, `42.`, `1.`, or `-3.5`. A float literal must have a decimal point in order to be counted as a float; a literal without a decimal point like 1 or 42 will be interpreted as an integer.

3.2.4.3 Boolean literals The boolean literal `true` is stored as a byte of value 1. The boolean literal `false` is stored as a byte of value of 0.

3.2.4.4 Character literals A character literal is a single character surrounded by single quotes ' ' and stored as a 1-byte ASCII value.

3.2.5 Delimiters

3.2.5.1 Parentheses Parentheses ( ) are used in function declaration and calling, and in expressions to modify operator precedence. Conditionals and loops also require parentheses.

3.2.5.2 Curly braces Curly braces { } are used to denote the start and end of a block of code; they are required after function declarations and at the beginning and end of each new block of code.

3.2.5.3 Matrix Bracket Bar Brackets with a bar symbol following it immediate are used to denote a matrix literal, and must be used for matrix assignment.

```
int[2][2] a;
a = [| [1, 2] & [3, 4] & [5, 6] & [7, 8] |];
```

3.2.5.4 Brackets Brackets [] are used for vector/matrix access and vector literals.

3.2.5.5 Semicolon A semicolon ; is required to terminate a statement. Any expression that is terminated with a semicolon will be executed on runtime. If this expression has no "side effect" such as assignment or function calling, then the compiler will throw an error. For loops will require semicolons in its syntax.

3.2.5.6 Commas Commas are used to separate expressions in function parameters and elements in list initialization.

3.2.5.7 Ampersand Ampersands & are used to separate vectors inside a matrix. E.g. [| [1,2,3] & [4,5,6] & [7,8,9] |]

3.2.5.8 Whitespace Whitespace is ignored in compilation and statements will be terminated only on semicolons, and not whitespace, tabs, or newlines. Each token in an expression is separated by these whitespace values.

3.3 Syntax and Semantics

3.3.1 Type specifiers

Types specifiers in declarations define the type of a variable or function declaration. To declare a variable, you must specify the type of the variable before its name, and to declare a function, you must specify its return type before its header. Examples of this can be seen below.

```
int x = 3; :) an integer of 3
def int myFunction(){} :) return an integer
```

3.3.2 Basic types

3.3.2.1 Integers pixelman supports integers through the `int` type. An example of declaring and initializing an `int` variable:

```
int i;
i = 42;
```

3.3.2.2 Floating points pixelman supports single-precision, 32-bit floating point numbers through the `float` type. It is possible to assign integer values to a `float` type. This will result in casting the integer to a float.

An example of declaring and initializing a `float` variable through a `float` literal and an `int` literal:

```
float f;
float f2;
f = 42.42;
f2 = -3
```

3.3.2.3 Booleans pixelman supports boolean values through the `boolean` type, and the `int` type. When evaluating integer values other than `true` and `false`, 0 evaluates to `false`, and any nonzero value evaluates to `true`. Trying to evaluate `null` as a `boolean` value results in an error. If a boolean evaluates to null, the compiler will throw an error. An example of declaring and initializing a `boolean`:

```
bool isColor;
bool isNonzero;
isColor = true;
isNoneZero = 1;
```

3.3.2.4 Characters pixelman supports chars which will be single characters surrounded by single quotes as seen in the examples below. Chars will be ASCII and have integer values ranging from 0-127.

```
char myLetter;
char myOtherLetter;
myLetter = 'a';
myOtherLetter = 'b';
```

3.3.2.5 Strings pixelman supports strings through the `string` type, which are stored as a `List` of `char` values.

```
string str;
str = "i am a string";
```

3.3.2.6 Void The `void` return type is only available to be used in functions that will not return any values.

```
def void functionName() {}
```

3.3.3 Vector

In pixelman, a `Vector` is an indexed container used to store multiple objects of the same type, with a static size and mutable elements. `Vector` elements are indexed beginning at position zero. To access the length of a `Vector`, perform the operation

```
sizeof(lst); :) this will return an int value of the length of vector lst.
sizeof(mat); :) this will return an int value of the number of rows of matrix mat.
```

3.3.3.1 Declaring Vectors A `Vector` may by declared by specifying the data type for its elements, the number of elements it can store, and its name. The number of elements must be a positive `int` value. For example, to instantiate a vector of integers of length 3:

```
int[3] rgb;
```

This `Vector` will initialize with every element defaulting to 0.

3.3.3.2 Initializing Vectors You can initialize the elements in a `Vector` when you declare it by enumerating the initializing values, separated by commas, in a set of brackets. Here is an example of creating a `Vector` of the `int` values [255, 0, 0]:

```
int[3] rgb;
rgb = [255, 0, 0];
```

When a `Vector` is initialized this way, all elements must be specified.

3.3.3.3 Accessing Vector Elements Vector elements will be accessed by writing the name of the list followed immediately by an open bracket, the number of the element, and a close bracket as seen in the example below.

```
int[3] rgb;
rgb = [255, 0, 100];
int x = rgb[1]; :)x now has the value 0.
```

3.3.4 Matrix

You can make a two-dimensional `Matrix`, or a "vector of vectors", by adding an extra set of square brackets and list length.

Our languages only work for 3 by 3 matrices. The reason is because most image convolution is done with 3 by 3 matrices.

```
int[3][3] mat;
mat = [| [42, 0, 0] & [0, 42, 0] & [0, 0, 42] |];
```

Matrix elements are accessed by specifying both row and column indices in brackets. Specifying only one bracket will return either the row vector or the column vector.

```
mat[1][1]; :) returns 42
mat[2][2]; :) returns 42
mat[0][]; :) returns [42,0,0]
mat[][0]; :) returns [0,0,42]
```

3.3.5 Type conversions

pixelman will implicitly convert integer to float if performing operations with mixed types. This means that any arithmetic operation with a float and an integer will return a float. The user may explicitly typecast on integers and floats, but may not explicitly convert vectors, matrices, strings, or characters.

To explicitly cast a type:

```
$type variable;

$int 3.5 :)returns 3
$int 3 :)returns 3
$float 3 :)returns 3.0
$float 3.1 :)returns 3.1
```

3.3.6 Operators

3.3.6.1 Arithmetic Addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`) work like regular arithmetic for types `int` and `float`. If applying any of these arithmetic operators on both an `int` and a `float`, the return type will be a `float`.

The modulus operator (`%`) returns the remainder after dividing two arguments. The two arguments must be integers.

Addition, subtraction, multiplication, and division are illegal on string, boolean, and image types.

3.3.6.2 Comparison The operator `==` is used to compare value of two operands of the same type. `==` is supported for `int`, `float`, and `char`. It does not allow for comparison between an `int` value and a `float` value. The operators `>`, `<`, `>=`, and `<=` are used to compare `int` values and `float` values, and also cannot compare values between the two.

```
bool a;
a = 42;
a == 42; :) evaluates to true
a == 41; :) evaluates to false
a == 42.0; :) compiler will throw error

a < 43; :) evaluates to true
a > 43; :) evaluates to false
a <= 42; :) evaluates to true
a >= 42; :) evaluates to true
```

3.3.6.3 Boolean The boolean operators `!`, `&&`, and `||` are supported for all types of operands.

```
bool a = ! true; :) evaluates to false
bool b = true && true; :) evaluates to true
bool c = true || false; :) evaluates to true
```

3.3.6.4 Bitwise The operators `<<` and `>>` can be used to bit shift ints left and right respectively.

```
int a = 4;
int b =  a << 2; :)evaluates to 16
int c = a >> 1 :)evaluates to 2
```

The operators `&`, `|`, and `^` can be used to represent the bitwise operations of and, or, and xor respectively.

```
int a = 4;
int b =  5;
int c = a & b :)evaluates to 4
int d = a | b :)evaluates to 5
int e = a ^ b :)evaluates to 1
```

3.3.6.5 Operator precedence Statements with multiple operators in pixelman will follow the order of operations. This means that statements in parenthesis will be evaluated first. Multiplication and division will have the next highest precedence. Finally, addition and subtraction will be evaluated last. As an example, the statement below will evaluate to 61.

```
(4 + 5 * 5) + 8 * 4  :) evaluates to 61
```

3.3.7 Matrix/vector operations

The operations `+`, `-`, and `*` are supported for vertices and matrices of type `int` or `float`.

In order to perform these operations on lists, their dimensions must work for matrix or vector multiplication. The following cases for multiplying vectors/matrices `A` and `B` are accepted:

- If `A` and `B` are both vectors of length `3`,
  - `A + B` will return an 3-length vector of the sum of each element in its corresponding index.
  - `A - B` will return an 3-length vector of the result of `A[i] - B[i]` in index `i`.
  - `A * B` will return a scalar inner dot product of the two lists.

  If one of `A` or `B` is of type `float`, then the return type will also be of type `float`; otherwise, the return type will be of type `int`.

- If `A` is a matrix of dimension `3` by `3` and `B` is a matrix of dimension `3` by `3`,
  - `A + B` and `A - B` returns the matrix sum/difference of the two matrices.
  - `A * B` will return an `3` by `3` matrix which is the matrix product of the two matrices.
- If one operand is a `int` or a `float` and the other is a vector or matrix of length `3`, scalar multiplication is performed on the vector/matrix, and the return type of the vector/matrix is a float if at least one of the operands is of float type(s).

The type of the output is defined as such:

- If at least one operand is of type `float` and one is of type `float` or type `int`, the return type will be of type `float`.
- If both operands are of type `int`, then the return type will be of type `int`.

3.3.8 Assignment

There are 11 assignment operators; all are syntactically right-associative (they group right-to-left). Thus, a=b=c means a=(b=c), which assigns the value of c to b and then assigns the value of b to a.

Operators:

```
= *= /= %= += -= <<= >>= &= ^= |=
```

The result of the first operand of an assignment operator must be a variable, or a compile-time error occurs (cannot do something like 3=6). This operand may be a named variable, such as a local variable or a field of the current object, or it may be a computed variable, as can result from a field access or an array access.
The type of the assignment expression is the type of the variable.

3.3.9 Functions

3.3.9.1 Declaration Functions are syntactically defined as:

```
def <type> <functionName>(arg1...argn) {
    <vdeclaration1>; <vdeclaration2>;....<vdeclaration_n>;
    <statement1>; <statement2>; ...; <statementn>;
    [return-statement;]
}
```

where `<type>` is the return type of the function, `functionName` is the unique label for a function, and `arg1...argn` are the optional arguments provided for the function.

3.3.9.2 Function calls To execute a function, it must be called correctly with its unique name and the required parameters.
Function calls are syntactically defined as:

```
<functionName>(arg1...argn);
```

If a function has the incorrect number of arguments or an unrecognizable format, the line will return a compiler error.

3.3.10 Names and scope

Variables defined in a block of code will only be accessible in that block of code. If a variable is created with a name that already exists in that block of code, the compiler will throw an error.

3.3.11 Built-in Library

pixelman supports various functions through its standard library:

3.3.11.1 void print_string will print to standard out the designated string.

3.3.11.2 void print_int(int i) will print to standard out the designated integer.

3.3.11.3 void print_float(float f) will print to standard out the designated float.

3.3.11.4 void print_newline() will print newline character to standard out.

3.3.11.5 void printb(bool b) will print to standard out the designated boolean.

3.3.11.6 int sizeof(Vector v) will get the length of the vector. On success, it will return the int value of the vector length.

3.3.11.7 float[3] vec_int_to_float(int[3] a) will convert vector a from type int to type float and return a.

3.3.11.8 int[3] vec_float_to_int(float[3] a) will convert vector a from type float to type int and return a.

3.3.11.9 float[3][3] mat_int_to_float(int[3][3] a) will convert matrix a from type int to type float and return a.

3.3.11.10 int[3][3] mat_float_to_int(float[3][3] a) will convert matrix a from type float to type int and return a.

3.3.11.11 int[3] scalar_mult_vec_(int a, int[3] b) will multiply all values in vector b by scalar a and return a vector with new values (type int).

3.3.11.12 float[3] scalar_mult_vec(int a, int[3] b) will multiply all values in vector b by scalar a and return a vector with new values (type float).

3.3.11.13 def int[3][3] mat_transposei(int[3][3] a) will return transpose of vector a(type int).

3.3.11.14 def float[3][3] mat_transposef(float[3][3] a) will return transpose of vector a(type float).

3.3.11.15 float det_mat2(float[2][2] a) will return the determinant of 2x2 matrix a.

3.3.11.16 float det_mat3(float[3][3] a) will return the determinant of 3x3 matrix a.

3.3.11.17 float[2][2] mat_inverse2(float[2][2] a) will return the inverse of 2x2 matrix a.

3.3.11.18 float[3][3] mat_inverse3(float[3][3] a) will return the inverse of 3x3 matrix a.

3.3.11.19 int[3] get_mat_rowi(int[3][3] a, int b) will return row b of 3x3 matrix a (type int).

3.3.11.20 float[3] get_mat_rowf(float[3][3] a, int b) will return row b of 3x3 matrix a (type float).

3.3.11.21 int[3] get_mat_coli(int[3][3] a, int b) will return columm b of 3x3 matrix a (type int).

3.3.11.22 float[3] get_mat_colf(float[3][3] a, int b) will return column b of 3x3 matrix a (type float).

3.3.11.23 void print_vecf(float[3] a) will print the vector a (type float).

3.3.11.24 void print_veci(int[3] a) will print the vector a (type int).

3.3.11.25 void print_matf(float[3][3] a) will print the matrix a in 3x3 format (type float).

3.3.11.26 void print_mati(int[3][3] a) will print the matrix a in 3x3 format (type int).

3.4 Statements

3.4.1 If.. else

pixelman supports `if/else` statements that allow conditional boolean statements to control the execution flow of the code. Conditional `if` statements can be nested multiple times. An `if` statement may be followed by an `else` that will execute if the condition in the if statement evaluates to false.

```
def <type> <functionName>(arg1...argn){
    if(<booleanExpression) {
        :) code
    } else {
        :) code
    }
}
```

3.4.2 Loops

3.4.2.1 For loops pixelman supports `for` loops that will run a block of code for as long as the conditional statement holds. The `for` loop will have a starting point `<start>`, typically an assignment to a variable, and that starting point will change according to the `<newAssignment>`. Once changed to `<newAssignment>`, the for loop will evaluate the `<conditionalStatement>` once again, and execute the function if true. Otherwise, it will exist the `for` loop. `for` loops may be nested multiple times.

```
def <type> <functionName>(arg1...argn){
    for(<start>; <booleanExpression>; <newAssignment>) {
        :) code
    }
}
```

3.4.2.2 While loop pixelman supports `while` loops that will run a block of code as long as the condition in the `while` evaluates to true.

```
def <type> <functionName>(arg1...argn){
    while(<booleanExpression>) {
        :) code
    }
}
```

3.4.3 Blocks

Blocks in pixelman will begin with a "{" and end with a "}." Blocks can be nested inside of each other. A "}" will mark the end of the block that began with the most recent "{" that has not yet been closed. Blocks can only be used following function declarations, `if` statements, `else` statements, `for` loop declarations, and `while` loop declarations.

3.4.4 Return

Return statements are defined as:

```
return <something>;
```

The return type must match the type explicitly stated in the function declaration. If it does not match the type, it will throw an error.

3.5 Formal Grammar

1. **Program**
   program:
   decls `EOF`

2. Declarations
   decls:
   decls vdecl
   decls fdecl

   fdecl:
   `DEF` *typ* `ID (` *formals_opt* `)` { *vdecl_list statement_list* }

   formals_opt:
   /* nothing */
   formals_list

   formals_list:
   typ `ID`
   formals_list `,` *typ* `ID`

   typ:
   `int`
   `float`
   `char`
   `bool`
   `string`
   `void`
   vec_t
   mat_t

   vdecl_list:
   /* nothing */
   vdecl vdecl_list

vdecl:

typ `ID ;`

vec_t:

typ `[` *expr* `]`

mat_t:

typ `[` *expr* `] [` *expr* `]`

3. **Statements**

stmt_list:

/* nothing */ *stmt_list stmt*

stmt:

expr `;`
`return ;`
`return` *expr* `;`
`{` *stmt_list* `}`
`if (` *expr* `)` *stmt*
`if (` *expr* `)` *stmt* `else {` *stmt* `}`
`while (` *expr* `)` *stmt*
`for` *(expr_opt; expr; expr_opt)* `{` *statement_list* `}`

4. Expressions

expr_opt:

/* nothing */
expr_list:

expr_list:

expr expr_list
expr

expr:

primary
unop expr
expr binop expr
expr asgnop expr

vector_literal:

expr `,` *vector_literal*
expr

matrix_literal:

`[` *vector_literal* `] &` *matrix_literal*
`[` *vector_literal* `]`

primary:

identifier
constant
string
`(` *expr* `)`
`ID [` *expr* `]`
`ID [` *expr* `] [` *expr* `]`
`ID [` *expr* `] [ ]`
`ID [] [` *expr* `]`
`[` *vector_literal* `]`
`[|` *matrix_literal* `|]`

`ID ( expr_list )`
`sizeof ( expr )`

The primary-expression operators
`()` `[]` `[|`
have highest priority and group left-to-right. The unary operators ! and - (negation) have priority below the primary operators but higher than any binary operator, and group right-to-left. The type-casting operators `$float` and `$int` have priority below arithmetic operators but above other binary operators. Binary operators all group left-to-right, and have priority decreasing as indicated:

binop:

`*` `/` `%`
`+` `-`
`<<` `>>`
`<` `>` `<=` `>=`
`==` `!=`
`&`
`^`
`|`
`&&`
`||`

unop:

`-` `!`
`$int` `$float`

The assignment operator has priority below all other binary operators, and groups right-to-left.

asgnop:

`=`

The comma operator has the lowest priority, and groups left-to-right.

3.6 Sample Program

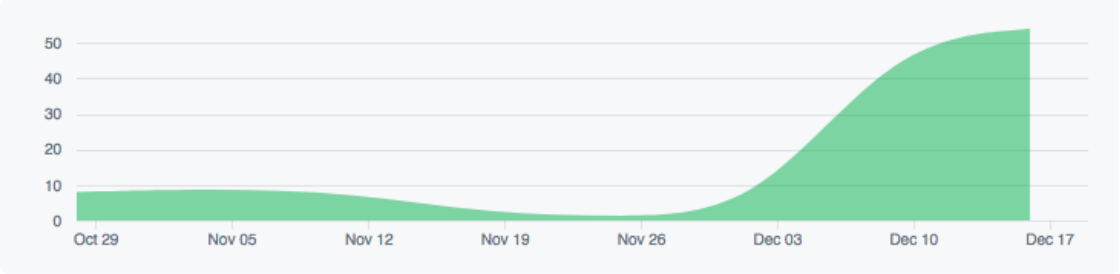
Included is a program that shows how a user can interact with matrices. The program creates a matrix and a vector and shows how to use many of the functions included in our standard library.

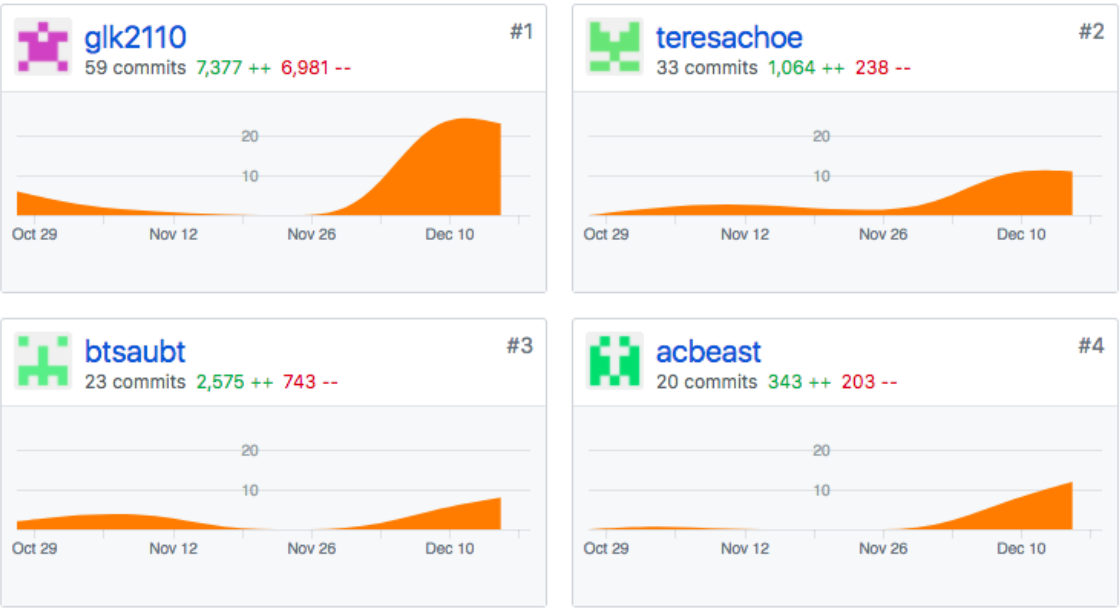
```
def int main(){
    int[3][3] a;
    int[3] b;
    int c;
    int d;
    float[3][3] e;
    c = 2;
    d = 1;
    a = [| [1, 2, 3] & [4, 5, 6] & [7, 8, 9] |];
    :) print_string("a = [| [1,2,3] & [4,5,6] & [7,8,9] |]");
    print_newline();

    print_string("array");
    print_newline();
    print_mati(a);
    print_newline();

    print_string("a[0][0] = ");
    :) print_newline();
    print_int(a[0][0]);
    print_newline();
    print_newline();

    print_string("c = 2, d = 1, a[c][d] = ");
```

```
    print_int(a[c][d]);
    print_newline();

    print_string("sizeof(a):");
    :) print_newline();
    print_int(sizeof(a));
    print_newline();

    b = a[0][];
    print_string("b = a[0][], sizeof(b):");
    :) print_newline();
    print_int(sizeof(b));
    print_newline();
    print_newline();

    print_string("print a:");
    print_newline();
    print_mati(a);
    print_newline();

    print_string("print transpose of a:");
    print_newline();
    print_mati(mat_transposei(a));
    print_newline();
    e = mat_inverse3(a);

    print_string("print inverse of a:");
    print_newline();
    print_matf(e);
    print_newline();

    return 0;
}
```

4 Project Plan

4.1 Process





4.1.1 Planning

Our group regularly met once a week on Sundays to work as a team, and group members occasionally went to office hours to meet with TAs for help with the project. We experienced some difficulty finding time to commit to working on the project, and at certain points in development of features, we experienced problems such as more than one member working on the same thing or dependencies not working. Our group also regularly met with Heather, our TA every Friday when she could meet. To help with bugs, we also emailed her and requested additional meetings with her (which was so nice of her!). To help with determining goals, we raised issues on the Github repository for pixelman and assigned them to members who were working on those components. It was also helpful to create objectives for each person or to partner off so that something would be accomplished at the end of the meeting. Towards the end of the project, we met every day, while also working remotely on the project.

4.1.2 Testing

All members of the group were expected to test their code and ensured it worked before pushing to master. When something was pushed onto master, we would check on our individual machine if the code broke any of the tests in our test suite. If it did, we would find the error, or as a last case scenario, roll back the code. We also focused on writing unit tests so we could pinpoint exactly where any errors appeared. After we implemented a new feature, a unit test for each part of that feature was added to the test suite. A full test for the entire feature was also added to the test suite to ensure it worked properly. Occasionally, we also added tests that we expected to fail to make sure they threw the correct errors.

4.2 Style Guide

We used underscores to separate words in variable names throughout our code. In the ast we used camel case. We tried to not have any lines of code greater than 100 characters long.

4.3 Project Timeline

Date	Measurable
9/27/2017	Project proposal submitted
10/16/2017	Language reference manual submitted
10/29/2017	Git repository initialized with Micro C skeleton code
11/17/2017	Hello world compiles and executes
11/29/2017	Grammar completed
12/13/2017	Matrices and vertices implemented
12/19/2017	Code generation
12/19/2017	Standard Library

4.4 Roles and Responsibility

Initially, we assigned the roles: Anthony - Manager, Gabe - Language Guru, Teresa - Tester, Brian - System Architect, but as the project progressed we ended up working on parts that suited our coding styles better. Since we worked mainly in the same room, everybody ended up touching most of the files, but members of the group eventually concentrated on individual files.

Team Member	Responsibility
Anthony Chan	Compiler Frontend, Semantic checking, Debugging, stdlib
Brian Tsau	Compiler Backend, Semantic checking (& SAST)
Gabriel Kramer-Garcia	Compiler Frontend, Linking, C Library, Test Suite
Teresa Choe	Compiler Frontend, Compiler Backend, Code Generation, stdlib

4.5 Development Environment

Github

We hosted our git repository on Github for version control, and used it to collaborate on the project and post issues that needed to be solved.

Ocaml 4.2.01

Our language was written in Ocaml 4.2.01, and pixelman programs were lexed through Ocamllex and parsed through Ocamlyacc.

LLVM 3.7

LLVM was the IR we compiled to in our language, which was then directly executed with an lli command.

4.6 Project Log

```
commit 6281a22a1bedc9351564448c4bfc33761d8413aa
Author: anthony chan <tn.chan5@gmail.com>
Date:   Wed Dec 20 16:21:43 2017 -0500

    fix stdlib. FINAL

commit 0849b4164137a15bda1172e00e1ebc1d140a2859
Author: acbeast <31133077+acbeast@users.noreply.github.com>
Date:   Wed Dec 20 15:55:48 2017 -0500

    add spaces to print_vecf/i

commit e52fc79683b291933e500f030b13c6ddd105723b
Merge: 484f7af e02d829
Author: anthony chan <tn.chan5@gmail.com>
```

```
16 Date:   Wed Dec 20 13:13:55 2017 -0500
17
18     Merge branch 'master' of https://github.com/btsaubt/pixelman
19
20 commit e02d8291c03efcefea421ff4dd2ff76c2b7995b6
21 Merge: 1e254d3 596f654
22 Author: Teresa Choe <tc2716@columbia.edu>
23 Date:   Wed Dec 20 13:13:53 2017 -0500
24
25     merge
26
27 commit 484f7af89c9a405bdcd3abfad359fe9af67de0c3
28 Merge: 03f7460 1e254d3
29 Author: anthony chan <tn.chan5@gmail.com>
30 Date:   Wed Dec 20 13:13:32 2017 -0500
31
32     Merge branch 'master' of https://github.com/btsaubt/pixelman
33
34 commit 03f7460f144655a4c53e576ae378ff2b714611a0
35 Author: anthony chan <tn.chan5@gmail.com>
36 Date:   Wed Dec 20 13:12:54 2017 -0500
37
38     fix vector.px demo file (add newlines), remove primitives.px
39
40 commit 596f65493b6db4aa148d7df7bc61f60c50e39c9e
41 Author: Teresa Choe <tc2716@columbia.edu>
42 Date:   Wed Dec 20 13:12:44 2017 -0500
43
44     pretty printing
45
46 commit 1e254d3a213ef092cd46460e57a31dc0972a04c2
47 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
48 Date:   Wed Dec 20 13:11:41 2017 -0500
49
50     tests
51
52 commit 23e23653c2e2fc4a6ab8b74b607b9b616e3d917e
53 Merge: d510d76 df8b903
54 Author: btsaubt <btsaubt>
55 Date:   Wed Dec 20 13:00:14 2017 -0500
56
57     Merge branch 'matrices'
58
59 commit df8b903e2ba7f38bcdb96ad09202e35d26ea2b35
60 Author: btsaubt <btsaubt>
61 Date:   Wed Dec 20 13:00:06 2017 -0500
62
63     added inverse * a (doesnt work right)
64
65 commit d510d767dab1362d7fc3674052eb7a2bcfd9e57c
66 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
67 Date:   Wed Dec 20 12:58:48 2017 -0500
68
69     fix tests
70
71 commit 921333e060621a7f5ef6d35c39e1f3e48cb02848
72 Merge: 32383b1 1c87de3
73 Author: anthony chan <tn.chan5@gmail.com>
74 Date:   Wed Dec 20 12:50:06 2017 -0500
75
76     Merge branch 'master' of https://github.com/btsaubt/pixelman
77
78 commit 32383b175cdc94991df3c589769b8b7aa1ab013c
```

```
Author: anthony chan <tn.chan5@gmail.com>
Date:   Wed Dec 20 12:48:40 2017 -0500

    fix all warnings and comment out image

commit 1c87de3e86a1a9f0f61eccb058e22342912c4c14
Merge: 31edfe9 414a063
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Wed Dec 20 12:46:09 2017 -0500

    merge

commit 31edfe9fc99611e27b257d7221cd84970fd6c56c
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Wed Dec 20 12:45:27 2017 -0500

    test for casts

commit 26b9d77720df1ab720878ba02d82420ea2573507
Merge: f1a724c 4afcd5a
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Wed Dec 20 12:32:00 2017 -0500

    merge conf1

commit f1a724cbd596d463d526356057bd1db72c13377d
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Wed Dec 20 12:31:23 2017 -0500

    changes to std

commit 19a62c07db92a9875fec8e566a40459b1e6d624a
Merge: f8cc077 921333e
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 11:41:23 2017 -0500

    Merge branch 'master' into matrices

commit f8cc0772edf7b3af84227ea1bac4c3bb4577b7f0
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 11:38:39 2017 -0500

    changed matrix sample program

commit 414a063fd81a7c01de86349749b8f733452e1362
Merge: b4dadd7 ebaadf9
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 11:29:30 2017 -0500

    Merge branch 'master' of https://github.com/btsaubt/pixelman

commit ebaadf9df7bf20cf30d83929090bc9e94e18bc04
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Wed Dec 20 12:42:49 2017 -0500

    add library function

commit b4dadd7b3a3f26d19e84e6dbdae5494f5cbf07cc
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 11:29:19 2017 -0500

    added print_newline()

```

```
commit 4afcd5a777bb06b725aaad77889de8a64651042e
Merge: abbf3e8 1c1515b
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 11:14:56 2017 -0500

    Merge branch 'master' into matrices

commit abbf3e85b14ae488ac7c406a571e1c84684ccf1a
Author: btsaubt <bt2420@columbia.edu>
Date:   Wed Dec 20 12:20:14 2017 -0500

    fixed transpose

commit 70f769167da8ac89b41326ec6a3aadcdecb5dda5
Author: btsaubt <bt2420@columbia.edu>
Date:   Wed Dec 20 12:18:19 2017 -0500

    got inverse for 3x3

commit 1c1515bbb6a38e8bb3ea547935157655bb89130d
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Wed Dec 20 11:25:47 2017 -0500

    fix tests and amke image demo

commit d79b658e9c8ed5a2c2e4df5e38398f1953955698
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 11:14:06 2017 -0500

    more work on stdlib, 3x3 inverse not working yet

commit e45afcacf2bbc31e547b9aa701d7fec5e3d89423
Merge: 5e00f4b 3990091
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 02:03:04 2017 -0500

    Merge branch 'master' into matrices

commit 5e00f4b60fdc1f16f1247776dc3bcac7b9869ab4
Author: btsaubt <btsaubt>
Date:   Wed Dec 20 02:02:43 2017 -0500

    added det and transpose to stdlib

commit 399009138e38d2e783233362524ae714d3ed649d
Author: anthony chan <tn.chan5@gmail.com>
Date:   Wed Dec 20 01:28:33 2017 -0500

    add primitives sample skeleton

commit 37026201b28c67823b90e90f1d2d9d97a406bb21
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Wed Dec 20 01:24:02 2017 -0500

    fix

commit e31accf44b14bfeeee869b80e46802e9972add5b
Author: anthony chan <tn.chan5@gmail.com>
Date:   Wed Dec 20 01:18:11 2017 -0500

    sample program skeleton

commit ac87fb3abf6de8dc3eb83ce8efde8a3646f73c1b
```

```
205 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
206 Date:   Wed Dec 20 01:14:05 2017 -0500
207
208     pics
209
210 commit f38bd18d63f4ad80b7cd06f11c19cadfa8bcc63b
211 Merge: 9cf1fad eb60756
212 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
213 Date:   Wed Dec 20 01:11:02 2017 -0500
214
215     Merge branch 'master' of https://github.com/btsaubt/pixelman
216
217 commit 9cf1fadbec043af5bfe78a4285566fa597cbf233
218 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
219 Date:   Wed Dec 20 01:10:54 2017 -0500
220
221     add input output methods
222
223 commit eb60756e1b28cba70c72dab4abf43aa3cbdff4e8
224 Merge: 67f5164 7da8504
225 Author: btsaubt <btsaubt>
226 Date:   Tue Dec 19 22:28:25 2017 -0500
227
228     Merge branch 'master' into matrices
229
230 commit 67f516470b948573e8c92dc995cf67e774a539d1
231 Author: btsaubt <btsaubt>
232 Date:   Tue Dec 19 22:28:09 2017 -0500
233
234     matrix/vecotr library functons aded, sum typos fixd
235
236 commit 7da85044dff8cf77004d8faf2520b29f23067741
237 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
238 Date:   Tue Dec 19 22:15:19 2017 -0500
239
240     fix tests
241
242 commit fd040b0dde97e78da1b1dec4d184661cd721993e
243 Merge: 0c29611 b548011
244 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
245 Date:   Tue Dec 19 22:14:50 2017 -0500
246
247     Merge branch 'master' of https://github.com/btsaubt/pixelman
248
249 commit b5480113a154523fdefa8d48fb685cf8c59076bb
250 Author: anthony chan <tn.chan5@gmail.com>
251 Date:   Tue Dec 19 22:14:23 2017 -0500
252
253     image access semant checking works
254
255 commit 0c29611f918fdaf745660577378aea084aba35b2
256 Merge: f17b3fe 48f8936
257 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
258 Date:   Tue Dec 19 22:11:17 2017 -0500
259
260     Merge branch 'master' into gabe
261
262 commit f17b3fe6c4136804e7a53f71b19d5c64185c0853
263 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
264 Date:   Tue Dec 19 22:09:20 2017 -0500
265
266     makePic
267
```

```
commit 38491220a02dc9ff74d520c09c35a3b200966fa9
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Tue Dec 19 21:55:34 2017 -0500

    image stuff

commit 48f8936cb608f7279c12f7094dd345d494e3974d
Author: anthony chan <tn.chan5@gmail.com>
Date:   Tue Dec 19 21:52:20 2017 -0500

    fix merge conflict ast

commit 0f1fa2e759ca0b90f8ae15d7d8dd51d413529dfa
Merge: 4a21cff ed9e1f1
Author: anthony chan <tn.chan5@gmail.com>
Date:   Tue Dec 19 21:49:31 2017 -0500

    image access parse, ast, sast

commit ed9e1f1e23ac3ec592b923a57facdddcda87d2fe
Author: anthony chan <tn.chan5@gmail.com>
Date:   Tue Dec 19 21:46:08 2017 -0500

    image access parse, ast, sast

commit 4a21cff1974c38b5c80c320a9a7358a8c53f1cb5
Merge: c438847 8d77a32
Author: btsaubt <btsaubt>
Date:   Tue Dec 19 21:03:26 2017 -0500

    Merge branch 'master' into matrices

commit c43884720b41563d53a7e68928b55ea5c7733f90
Author: btsaubt <btsaubt>
Date:   Tue Dec 19 21:02:43 2017 -0500

    Added matrix row/col access in grammar and semant, need to write
        function calls in library

commit 8d77a32a94aea717244db6659813bd5ffe8495bc
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Tue Dec 19 20:33:20 2017 -0500

    added std lib

commit b8bfdcc230a481738465ac6cab99b812e2adb270
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Tue Dec 19 19:25:26 2017 -0500

    makePIc

commit d69827e84891ca83917668d3e8f80410a7608074
Author: btsaubt <bt2420@columbia.edu>
Date:   Tue Dec 19 15:48:47 2017 -0500

    started adding row/col access for matrices

commit efe8aa7092075d1d3a2b85d3afd2734b6ae7d7f1
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Tue Dec 19 00:38:19 2017 -0500

    added stdlib functions to a test file for now

```

```
commit 09a50f559f749dc8f8a9fa2730e3e862a7699f58
Merge: 29ed981 0bcc2d0
Author: btsaubt <btsaubt>
Date:   Tue Dec 19 00:30:50 2017 -0500

    Merge branch 'master' into matrices

commit 0bcc2d0fd7e571ee347ea5b1fe8f6dae64d108ac
Merge: 5fda6df a1206c3
Author: btsaubt <btsaubt>
Date:   Tue Dec 19 00:30:27 2017 -0500

    merge length into master

commit 5fda6df08020ef84cf0c16f244460c7dce614162
Author: btsaubt <btsaubt>
Date:   Tue Dec 19 00:25:28 2017 -0500

    got length working for vec

commit 29ed98142f88523301571ba9e3b7fcc9f4cdf649
Author: btsaubt <btsaubt>
Date:   Tue Dec 19 00:13:00 2017 -0500

    work on vector matrix type of variable length in actuals

commit 30e5d308b0b19339f39ab59f12b5202e9b3930ac
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Tue Dec 19 00:08:48 2017 -0500

    changes

commit 5103939296b62adcdb1e2748d96d5503570b9861
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 23:25:48 2017 -0500

    got rid of warnings

commit a1206c31b532d56a10eaf6abced0e25a259b3049
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 23:20:31 2017 -0500

    removed pointers for vec and mat

commit 7fa471e3b56bef346d52537488d152d01b8b9225
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 23:13:58 2017 -0500

    more stuff added to array length; it's not returning the right size

commit 29088f25ed12debfd7cd206b65da8b20197e4dd2
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 22:50:11 2017 -0500

    fixed some warnings

commit 8d7308f961421c3682be133d8ea0efebb8372799
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 22:41:53 2017 -0500

    fixed stupid type checking error yuck

commit 3445bd077579c02593b0db551c3d17c8ed4baac7
```

```
393 Author: btsaubt <bt2420@columbia.edu>
394 Date:   Mon Dec 18 21:54:05 2017 -0500
395
396     test for vector addition
397
398 commit 4c27d827d90c629c06bc33cbad66eef147740a3b
399 Author: btsaubt <bt2420@columbia.edu>
400 Date:   Mon Dec 18 21:53:52 2017 -0500
401
402     wrok on operator overloading
403
404 commit 9400e59d53155e72e37d6f3902e781dc962baee6
405 Author: Anthony Chan <tn.chan5@gmail.com>
406 Date:   Mon Dec 18 20:30:29 2017 -0500
407
408     attempt at vector/matrix length
409
410 commit 3619d13336c96c234ed4e33f5bac74bc0bafc8ce
411 Merge: f2e1ca8 8277738
412 Author: btsaubt <bt2420@columbia.edu>
413 Date:   Mon Dec 18 14:29:46 2017 -0500
414
415     Merge branch 'master' into matrices
416
417 commit 82777382dce8b3f0904a1f066d1fd0baf4006f5f
418 Author: Teresa Choe <tc2716@columbia.edu>
419 Date:   Mon Dec 18 14:29:07 2017 -0500
420
421     Added matrix/vector operators to semant and some tests
422
423 commit f2e1ca82300114e0be421fd6d8007504f2d1adcb
424 Author: btsaubt <bt2420@columbia.edu>
425 Date:   Mon Dec 18 14:28:44 2017 -0500
426
427     added int -> float type inferencing in assignment and function
          actuals
428
429 commit 2abd81131762909628efbdad49021aff4e000a71
430 Merge: 44f6ea8 642590a
431 Author: Teresa Choe <tc2716@columbia.edu>
432 Date:   Mon Dec 18 13:33:31 2017 -0500
433
434     merge conflicts
435
436 commit 642590ae4496294c7296bebecd7f0e5df0c46a31
437 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
438 Date:   Mon Dec 18 13:32:36 2017 -0500
439
440     add loops back to tests
441
442 commit 5cb8d769e0425768f9b90c6d939a3e12c8b3c283
443 Author: btsaubt <bt2420@columbia.edu>
444 Date:   Mon Dec 18 13:26:15 2017 -0500
445
446     fixed basic block for matrix/vec access
447
448 commit 44f6ea87344dde04b4188a5b454f0ee634de7af7
449 Merge: cb6f379 bb7cc46
450 Author: Teresa Choe <tc2716@columbia.edu>
451 Date:   Mon Dec 18 13:22:19 2017 -0500
452
453     merge conflicts
454
```

```
commit cb6f379eeef12cce139d6501ba756c5b18b62a37
Merge: f30d29f d7276e0
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Mon Dec 18 13:19:02 2017 -0500

    merge conflicts

commit bb7cc46f69b05f06d7374a7033ed5201be961630
Merge: 926b3f4 d7276e0
Author: btsaubt <bt2420@columbia.edu>
Date:   Mon Dec 18 13:16:25 2017 -0500

    Merge branch 'master' into matrices

commit d7276e0305e66f1f6550326b438a52929ebe7da9
Merge: 2d36ec3 657cda8
Author: btsaubt <bt2420@columbia.edu>
Date:   Mon Dec 18 13:16:03 2017 -0500

    Merge branch 'master' of https://github.com/btsaubt/pixelman

commit 926b3f43b7906ec8b7c4ab92cf67a8aa5605aa91
Author: btsaubt <bt2420@columbia.edu>
Date:   Mon Dec 18 13:15:51 2017 -0500

    fixed error with matrix/vector literals, some work on operator
        overloading and matrix/vector pointers

commit 657cda8288c3085a01368be7eee2ab86e696f9e0
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Mon Dec 18 13:14:45 2017 -0500

    fix failing tests

commit ec2909be8ae7de2dc843b17e9d2c8e82c45d1bfc
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Mon Dec 18 12:51:20 2017 -0500

    fix cast tests

commit f30d29f6695e58c30d6dc90e20d9ce843feef1fb
Merge: c86c437 2d36ec3
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Mon Dec 18 01:24:00 2017 -0500

    merge conflicts

commit c86c43761f1a55ebbe6b523edf7d5bd27cf6156e
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Mon Dec 18 01:23:31 2017 -0500

    added multiline comments yay

commit 2d36ec3c5cd08faf07a404ad0079e9ac9536bdbf
Merge: ffc431c f9e9b5a
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 01:22:06 2017 -0500

    fixed merge conflict

commit ffc431c8b4fae9963229ba02c29663350b1fd8c7
Author: btsaubt <btsaubt>
Date:   Mon Dec 18 01:15:38 2017 -0500
```

```
517
518     added pointers to grammar, started matrix operator overloading
519
520 commit f9e9b5a953d747b7d3b36ac6dd509e3707250c6a
521 Author: Teresa Choe <tc2716@columbia.edu>
522 Date:   Mon Dec 18 01:08:31 2017 -0500
523
524     added matrix literal to codegen
525
526 commit 6fd84e5f94f104ac1932f252381eaaec01bcf67d
527 Author: btsaubt <btsaubt>
528 Date:   Sun Dec 17 23:32:08 2017 -0500
529
530     added vectors/matrices to types
531
532 commit 2e2b4e3373901b0d9fa1e230c8a8b5ada8bc46d0
533 Author: btsaubt <btsaubt>
534 Date:   Sun Dec 17 22:39:19 2017 -0500
535
536     tests for vector element access and assignment
537
538 commit 5ffd8a44a99f1a3cfcab3af67293dbd138efa603
539 Merge: 76cc926 06377b4
540 Author: btsaubt <btsaubt>
541 Date:   Sun Dec 17 22:35:39 2017 -0500
542
543     fixed merge conflicts
544
545 commit 76cc926dbf99e73d1ee2625c5d107c4434bafcfb
546 Author: btsaubt <btsaubt>
547 Date:   Sun Dec 17 22:34:46 2017 -0500
548
549     got vector/matrix access to make!
550
551 commit 06377b43fb5af810cb51243ba603be15e403f2f8
552 Merge: a9550e2 471dcff
553 Author: Teresa Choe <tc2716@columbia.edu>
554 Date:   Sun Dec 17 20:27:32 2017 -0500
555
556     merge conflict
557
558 commit a9550e2731469aad1ea7b542f39983be04eedb4f
559 Author: Teresa Choe <tc2716@columbia.edu>
560 Date:   Sun Dec 17 20:27:10 2017 -0500
561
562     Added bitwise operators.
563
564 commit 471dcffba7e71398f7167002417724 65b7750e5e
565 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
566 Date:   Sun Dec 17 19:50:43 2017 -0500
567
568     added cast tests
569
570 commit 699bcc7ee28c30476df062d82bed301a95e172b6
571 Merge: 790fa6f 810cb29
572 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
573 Date:   Sun Dec 17 19:43:50 2017 -0500
574
575     Merge branch 'master' of https://github.com/btsaubt/pixelman
576
577 commit 810cb2974740930d3187fc72fb74f1977d032b82
578 Author: btsaubt <btsaubt>
579 Date:   Sun Dec 17 19:42:52 2017 -0500
```

```
580
581     removed stuff from stdlib
582
583 commit 790fa6f382c0fed119876ce976b2858257220cf6
584 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
585 Date:   Sun Dec 17 19:35:35 2017 -0500
586
587     more tests
588
589 commit 1c54ffaa984b3c875b1548ce13095f27bdfb3274
590 Author: btsaubt <btsaubt>
591 Date:   Sun Dec 17 19:33:51 2017 -0500
592
593     moved parser back
594
595 commit 85846a530235120b8decda7cd2aabafd20011584
596 Merge: 9961b9e 5c059a2
597 Author: btsaubt <btsaubt>
598 Date:   Sun Dec 17 19:30:27 2017 -0500
599
600     Merge branch 'master' into matrices
601
602 commit 9961b9e391a52279909e620c60f2d88d54dd7ac8
603 Merge: 7ab547a 99e2129
604 Author: btsaubt <btsaubt>
605 Date:   Sun Dec 17 19:29:37 2017 -0500
606
607     fixed merge conflict
608
609 commit 5c059a2868a878e8d2781733634ddc8ee56aa4aa
610 Merge: 0c49ab5 99e2129
611 Author: Teresa Choe <tc2716@columbia.edu>
612 Date:   Sun Dec 17 19:26:47 2017 -0500
613
614     merge conflict
615
616 commit 0c49ab568b71c2ace41cfb6952c9ae2ddb6ee4b9
617 Author: Teresa Choe <tc2716@columbia.edu>
618 Date:   Sun Dec 17 19:26:27 2017 -0500
619
620     vector literal in code gen and added test
621
622 commit 7ab547ab3e291742113842709de38b938d3af235
623 Author: btsaubt <btsaubt>
624 Date:   Sun Dec 17 19:23:15 2017 -0500
625
626     fixed shift/reduce for casting
627
628 commit 99e2129d4eeae978c24e22f537657ae1b2d38e89
629 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
630 Date:   Sun Dec 17 19:16:20 2017 -0500
631
632     fix matrix tests
633
634 commit b1bc802a41509d19693d29b9dc40d08907c88ee9
635 Merge: 554ba68 09b7520
636 Author: anthony chan <tn.chan5@gmail.com>
637 Date:   Sun Dec 17 19:11:32 2017 -0500
638
639     Merge branch 'master' of https://github.com/btsaubt/pixelman
640
641 commit 554ba68b1508f683a794dc1b97a7ef9a312322ac
642 Author: anthony chan <tn.chan5@gmail.com>
```

```
643 Date:   Sun Dec 17 19:10:41 2017 -0500
644
645     Matrix semant works
646
647 commit 09b75200a1f135ab34858283ef3359e6239cd7d8
648 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
649 Date:   Sun Dec 17 19:07:06 2017 -0500
650
651     changed print to print_int and corresponding tests
652
653 commit 809be70dc523c204d3139c044009e50cdabb1680
654 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
655 Date:   Sun Dec 17 18:57:05 2017 -0500
656
657     another test
658
659 commit ab057e02c3169188fafcbc759081605c99715855
660 Merge: fb3dfe4 f9878f7
661 Author: btsaubt <btsaubt>
662 Date:   Sun Dec 17 18:52:30 2017 -0500
663
664     Merge branch 'master' into matrices
665
666 commit fb3dfe4a80da53c9cd466a9a11197b730acc1420
667 Author: btsaubt <btsaubt>
668 Date:   Sun Dec 17 18:49:40 2017 -0500
669
670     added type casting to grammar and semant
671
672 commit 1130c7448fedc5198730050917509eac0da075f9
673 Merge: 075ad6b f9878f7
674 Author: anthony chan <tn.chan5@gmail.com>
675 Date:   Sun Dec 17 18:49:17 2017 -0500
676
677     Merge branch 'master' of https://github.com/btsaubt/pixelman
678
679 commit 075ad6b7a5b204018a2be41af6f76c377ada1668
680 Merge: 396839c 3246a7d
681 Author: anthony chan <tn.chan5@gmail.com>
682 Date:   Sun Dec 17 18:40:37 2017 -0500
683
684     Merge branch 'vector_semant'
685
686 commit 396839c6b5515d1bc39d4286168cff1561afcf89
687 Merge: 2927c8e de0a27d
688 Author: anthony chan <tn.chan5@gmail.com>
689 Date:   Sun Dec 17 18:40:20 2017 -0500
690
691     Merge branch 'master' of https://github.com/btsaubt/pixelman
692
693 commit 3246a7d66e7d787982cc398e72134cde33b268fa
694 Author: anthony chan <tn.chan5@gmail.com>
695 Date:   Sun Dec 17 18:40:02 2017 -0500
696
697     Vector semant checking functional
698
699 commit f9878f72ccfe99d0a9b690c2947092910e3dda1b
700 Merge: c771ae0 de0a27d
701 Author: Teresa Choe <tc2716@columbia.edu>
702 Date:   Sun Dec 17 18:38:11 2017 -0500
703
704     merge conflict
705
```

```
commit c771ae0b0f226966e926788deb8674791a0a51ec
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Sun Dec 17 18:37:19 2017 -0500

    Changed grammar of matrices

commit de0a27db14c3b1f2c55e63916c732bcdb7475b99
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sun Dec 17 18:17:40 2017 -0500

    add unit tests

commit de4faf0e793b77a38954fdd5c8e3089406f8009c
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sun Dec 17 18:07:39 2017 -0500

    fixed fail tests

commit 2927c8e76ac802ccbe937d950e7ab4cdb9569f95
Merge: 2d4bacb f81000a
Author: anthony chan <tn.chan5@gmail.com>
Date:   Sun Dec 17 17:00:33 2017 -0500

    Merge branch 'master' of https://github.com/btsaubt/pixelman

commit f81000a65a6c12fef1b9d044503741267f427d9a
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sun Dec 17 16:37:01 2017 -0500

    fix parse error

commit b80a7305cf6a986a84f5b83885372d4342a8d798
Author: btsaubt <btsaubt>
Date:   Sun Dec 17 00:17:08 2017 -0500

    added type casting for ints and floats to grammar

commit d2d481751648ae136627e7f97e425f74acd3020a
Author: btsaubt <btsaubt>
Date:   Sat Dec 16 23:53:37 2017 -0500

    commented out stdlib functions, changed make clean to remove err

commit 3f2f8b4426475e2b00acf7b9460308608fa1a943
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sat Dec 16 18:17:24 2017 -0500

    added greyscale and sepia

commit 53fe8cf0cbe53628f9442cf6d1bb547bfe65db82
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sat Dec 16 18:03:32 2017 -0500

    add print_string

commit 2d4bacbc59195e45776dd081c76a0b5f61b52463
Author: anthony chan <tn.chan5@gmail.com>
Date:   Sat Dec 16 17:18:45 2017 -0500

    fix -no-pie

commit 970614d00a6920ab3e95ad03d5b8ef9c3f407d65
Merge: 50dfb30 c2c6430
```

```
Author: btsaubt <btsaubt>
Date:   Sat Dec 16 16:13:08 2017 -0500

    Merge branch 'master' into matrices

commit 50dfb3092ce5021cac31e922105b36d14736fff6
Author: btsaubt <btsaubt>
Date:   Sat Dec 16 16:12:46 2017 -0500

    finished vector/matrix literals in semant

commit c2c64306e1ee2703121ff91f4874662750cece96
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sat Dec 16 16:05:38 2017 -0500

    add greyscale

commit c371c98f2bbdd156ac0cd6354274e1ae9a18f6be
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sat Dec 16 14:56:57 2017 -0500

    added printb

commit 35d2b9558c331c5d8cab9638212794632cdc91fc
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Sat Dec 16 14:45:59 2017 -0500

    added print_string to protexted fucntions

commit 3fd072dfe96bbd82812aa209a7f5d37138bcca6b
Merge: be7b27c 067c7cf
Author: btsaubt <btsaubt>
Date:   Sat Dec 16 00:02:28 2017 -0500

    fixed merge conflicts

commit 067c7cfb2ddd0a82317cac131e153e2888a6c30a
Author: Anthony Chan <tn.chan5@gmail.com>
Date:   Sat Dec 16 12:21:16 2017 -0500

    fix unary and binary minus differentiation

commit be7b27c684a7dc408134babda6ff20f074f9be44
Merge: 54579ad fda3976
Author: btsaubt <btsaubt>
Date:   Sat Dec 16 00:00:45 2017 -0500

    fixed merge conflict

commit fda39766eb1bab5a00a0f6f208b180fff00a1c75
Author: btsaubt <bt2420@columbia.edu>
Date:   Sat Dec 16 13:46:24 2017 -0500

    more work on matrix/vec literals, type should be worked on next

commit 54579ad8eae79b9eaf5f38e055807691a34944fb
Author: btsaubt <btsaubt>
Date:   Fri Dec 15 23:59:46 2017 -0500

    started matrix and vector initialization in codegen

commit a502482ad0343f954acb27d031fb0f5c5ec2af00
Author: btsaubt <bt2420@columbia.edu>
```

```
832 Date:   Fri Dec 15 23:04:35 2017 -0500
833
834     added vec/mat init in codegen (not tested yet), and started vec/mat
            literals in semant
835
836 commit 4f55b8dd469616d1b2db1e4a03cc8ed035bcb317
837 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
838 Date:   Fri Dec 15 17:58:57 2017 -0500
839
840     connected stdlibgit statusgit status and wrote printb
841
842 commit e46cc436b2b124d6573016584053b89391f85734
843 Merge: 95d40c4 a06c926
844 Author: btsaubt <btsaubt>
845 Date:   Fri Dec 15 16:28:24 2017 -0500
846
847     Merge branch 'matrices'
848
849 commit a06c926e5e7f007e28d6aa3c2b6979165fcdbec4
850 Author: btsaubt <btsaubt>
851 Date:   Fri Dec 15 16:28:01 2017 -0500
852
853     fixed errors caused by merge conflict
854
855 commit 95d40c412f0affadee851b8ad087c28937f86778
856 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
857 Date:   Fri Dec 15 16:19:50 2017 -0500
858
859     refix tests
860
861 commit 74df167a533403ae6cbd84436d22e72cf152bc6d
862 Merge: c1e4a5f cae7c5a
863 Author: anthony chan <tn.chan5@gmail.com>
864 Date:   Fri Dec 15 15:51:17 2017 -0500
865
866     Merge branch 'master' of https://github.com/btsaubt/pixelman
867
868 commit c1e4a5f9216dc56a39b01e98a4cc3531fdc4ed0c
869 Author: anthony chan <tn.chan5@gmail.com>
870 Date:   Fri Dec 15 15:50:40 2017 -0500
871
872     Add printb back for printing bools
873
874 commit cae7c5a7831920098d454048cfd37d1775f280e9
875 Merge: e4db2bb 6da125f
876 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
877 Date:   Fri Dec 15 15:44:31 2017 -0500
878
879     Merge branch 'stdlib'
880
881 commit 6da125fd2c9ca661d0761ce4a567f048a11ecba7
882 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
883 Date:   Fri Dec 15 15:42:43 2017 -0500
884
885     test stdlib
886
887 commit e4db2bb13e48dd472f6f4e9312d90a3c28d66ca2
888 Merge: cc00d0c bcd41e9
889 Author: Teresa Choe <tc2716@columbia.edu>
890 Date:   Fri Dec 15 15:22:15 2017 -0500
891
892     merge conflicts
893
```

```
commit cc00d0ca6118d877cc8b35b9efe9422586cc0b77
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Fri Dec 15 15:02:52 2017 -0500

    test to check vector and matrix literals

commit 508792941e6e9a4c5b41bed053f275b24d2f298a
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Fri Dec 15 15:02:27 2017 -0500

    Matrix literals done

commit bcd41e95e6feab26a59dc08bca5b0ddac0177baa
Merge: befc540 3a64861
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Fri Dec 15 14:53:46 2017 -0500

    Merge branch 'master' into sast

commit befc540dfba5a73ce13905219a6d6a5fc1e7ec32
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Fri Dec 15 14:48:18 2017 -0500

    edit pixelman file

commit 3a648612f4e6dc79419b9ba1d79b82d7938ffc76
Merge: c262faf d6ee31f
Author: anthony chan <tn.chan5@gmail.com>
Date:   Fri Dec 15 14:43:08 2017 -0500

    Merge branch 'master' of https://github.com/btsaubt/pixelman

commit c262faf8a75efa8bb467808d8067765004c08a48
Author: anthony chan <tn.chan5@gmail.com>
Date:   Fri Dec 15 14:42:06 2017 -0500

    Change ast to sast and fix codegen for sast

commit fc6ca7cfff852fc41adf73e68ef7d13deee895c8
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Fri Dec 15 14:36:57 2017 -0500

    added literals for vectors

commit d6ee31f0ff85af3127946ccede1da7bef8d04511
Author: btsaubt <btsaubt>
Date:   Fri Dec 15 14:14:11 2017 -0500

    added vector/matrix access in grammar and semant

commit 597117900bab5d1be248f3bb387edc8a56f9bd78
Author: btsaubt <btsaubt>
Date:   Fri Dec 15 13:46:13 2017 -0500

    added checking vector/matrix dimensions for int in semant

commit 0d315e5a2c86c46ceee66c2dd8264f9667b27696
Merge: 3e6955f 12eda89
Author: btsaubt <btsaubt>
Date:   Fri Dec 15 13:37:15 2017 -0500

    Merge branch 'master' into matrices
```

```
commit 3e6955f48be23487dc6f855a644463fceea1c411
Author: btsaubt <btsaubt>
Date:   Fri Dec 15 13:36:58 2017 -0500

    finished semant to sast

commit f408cef1309061b9ac0e7a6d9f0c7a74ca6e8633
Author: btsaubt <bt2420@columbia.edu>
Date:   Thu Dec 14 22:34:14 2017 -0500

    completed semant ast -> sast, still errors

commit a4a56eccca1abe32ff6e3e7a4bbcedde0efcaa31
Author: btsaubt <btsaubt>
Date:   Thu Dec 14 18:17:16 2017 -0500

    more work on semant; some work done on stmts; sstmts not finished
        yet

commit 12eda893a72bc98d516cd2a51696e824869b4145
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 18:09:01 2017 -0500

    more tests

commit a1a5e0f1a2c55639fd6f108152dd6cb15092cf6b
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 18:05:17 2017 -0500

    tests

commit 85849355ca346ab31e46f74cee899aea4efef936
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 17:58:25 2017 -0500

    added more float tests

commit 0c46c214ed69c128d3d705e5c6e6ecd94d5cdd77
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 17:42:07 2017 -0500

    changed texts to work with print_float

commit bcb6ecb27eea91f30f0922b6638da8617581795b
Author: anthony chan <tn.chan5@gmail.com>
Date:   Thu Dec 14 17:28:46 2017 -0500

    fix printing float and float operations (change float to double)

commit dac674519d3a8749718911d0014f630eae0e3195
Author: anthony chan <tn.chan5@gmail.com>
Date:   Thu Dec 14 17:10:47 2017 -0500

    fix print float and float operations (change to double)

commit b5cff257174fe2b1dd6b0eca8398ee5e5118bf8c
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 16:37:52 2017 -0500

    more tests

commit dabd028578c90753c70e570d3a05dfbf06780e2a
Author: Teresa Choe <tc2716@columbia.edu>
```

```
Date:   Thu Dec 14 16:17:35 2017 -0500

    Added exception handling to operations to get rid of warnings

commit 83d468e96a5a53fe769df2098c59702697ee55de
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 16:04:00 2017 -0500

    fix fail tests

commit d552439a0edfb1f66e768fab66832778d05db7f8
Author: btsaubt <btsaubt>
Date:   Thu Dec 14 15:54:40 2017 -0500

    more changes to semant to convert ast to sast

commit 004471290cc479a9a7b1b0cd33696e14cc852482
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 15:53:56 2017 -0500

    remove err files

commit c6f7d74db370651bdb6767fc96db51642e4d69a2
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 15:53:14 2017 -0500

    fix test files

commit 85ea7d94a923bd635eea3407d35d99acc0269ff6
Merge: 6d222d7 c15dc32
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Thu Dec 14 15:48:43 2017 -0500

    fixed merge conflicts.

commit 6d222d7a14352ea7ed40ce0c095a8ecc22fd3611
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Thu Dec 14 15:47:43 2017 -0500

    fixed greater than for integers

commit c15dc328cc3915e15dbdb39f336f26fe920baa77
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 15:46:10 2017 -0500

    changed fail tests to .px

commit 13d5a282810119a9f1182fb57e4dda4bd8428e51
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 15:40:39 2017 -0500

    fix tests

commit 8ed54db0ec8bd08b656ba0ad3d055f2a17d105cf
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 15:07:22 2017 -0500

    change to our test suite

commit 3d0bcc867294fec9d170088b4d5c616e10e10acd
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Thu Dec 14 14:16:42 2017 -0500
```

```
1082      renamed everything
1083
1084  commit 7b3f2fca49a34d463aa5de3ce62014a225654e39
1085  Author: btsaubt <bt2420@columbia.edu>
1086  Date:   Thu Dec 14 13:16:49 2017 -0500
1087
1088      Added to semant, minor changes to all to generate sast from sast
1089
1090  commit 3f5301f1a519884a8451d0e53e72ee6653ad5899
1091  Author: btsaubt <bt2420@columbia.edu>
1092  Date:   Wed Dec 13 22:33:08 2017 -0500
1093
1094      changes to semant
1095
1096  commit 013deac94580d745e52e77f538e4e2c4f86cfded
1097  Merge: 2c9f56e 132cc5a
1098  Author: btsaubt <btsaubt>
1099  Date:   Wed Dec 13 21:00:06 2017 -0500
1100
1101      Merge branch 'master' into matrices
1102
1103  commit 2c9f56e5666f767798e8b89242bc98cd156112ec
1104  Author: btsaubt <btsaubt>
1105  Date:   Wed Dec 13 20:59:42 2017 -0500
1106
1107      added sast (not implemented yet)
1108
1109  commit 132cc5a05fc5c214b59382a42be7c98e1b184ad8
1110  Author: Teresa Choe <tc2716@columbia.edu>
1111  Date:   Wed Dec 13 20:53:31 2017 -0500
1112
1113      Forgot to add float test for ast, and it also compiles.
1114
1115  commit 523ba6f4e1d21d91c7046092525c59ab4dceeea9
1116  Author: Teresa Choe <tc2716@columbia.edu>
1117  Date:   Wed Dec 13 20:52:06 2017 -0500
1118
1119      Fixed floating pt regex.
1120
1121  commit 9476b893490f9a3e5a4bd7f9ba0637adc3c4a57e
1122  Merge: 63e7490 5813238
1123  Author: Teresa Choe <tc2716@columbia.edu>
1124  Date:   Wed Dec 13 20:37:13 2017 -0500
1125
1126      merge conflicts
1127
1128  commit 63e74905075c318a84c962819e2299b8bb7cdebf
1129  Author: Teresa Choe <tc2716@columbia.edu>
1130  Date:   Wed Dec 13 20:35:49 2017 -0500
1131
1132      Float operators, bitshift, and tests.
1133
1134  commit 58132383fed65411600be921504fe283a704c653
1135  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1136  Date:   Wed Dec 13 19:03:20 2017 -0500
1137
1138      update test
1139
1140  commit 35536cff2bdbab7a6d5a341eed5b7666ea7eee88
1141  Merge: 193aebf 774770e
1142  Author: anthony chan <tn.chan5@gmail.com>
1143  Date:   Wed Dec 13 19:01:20 2017 -0500
1144
```

```
1145      Merge branch 'master' of https://github.com/btsaubt/pixelman
1146
1147  commit 193aebfc2699e3a7e29024b086dcbd3c73e23890
1148  Author: anthony chan <tn.chan5@gmail.com>
1149  Date:   Wed Dec 13 18:59:33 2017 -0500
1150
1151      Fix print_string to not print quotes
1152
1153  commit 143e7213c5ce541e653b5484826a1841fb568d2a
1154  Author: Teresa Choe <tc2716@columbia.edu>
1155  Date:   Wed Dec 13 17:48:23 2017 -0500
1156
1157      Added shift right/left bitwise.
1158
1159  commit 774770efa020a188eeade909d64825a71e5c59d0
1160  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1161  Date:   Wed Dec 13 17:43:35 2017 -0500
1162
1163      fix test
1164
1165  commit b723b2a99af17e7535b323a2197dfe3518c0e3e6
1166  Merge: 307084c 8913c1c
1167  Author: Teresa Choe <tc2716@columbia.edu>
1168  Date:   Wed Dec 13 17:15:30 2017 -0500
1169
1170      fixed merge conflicts
1171
1172  commit 307084c69e22b5f3e56b0136ab4ee876570cd16f
1173  Author: Teresa Choe <tc2716@columbia.edu>
1174  Date:   Wed Dec 13 17:14:53 2017 -0500
1175
1176      Added modulus
1177
1178  commit 8913c1c1fbcdfdfa87cb1cfd64c536212779ad63
1179  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1180  Date:   Wed Dec 13 17:04:01 2017 -0500
1181
1182      back to ast
1183
1184  commit 7a91d3e087940bc39bfa4319594095e2b7305eeb
1185  Merge: 5977da2 cd79bab
1186  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1187  Date:   Wed Dec 13 16:48:35 2017 -0500
1188
1189      Merge branch 'master' into tests
1190
1191  commit cd79babf2afb3f8a3a98959dfbe68ed73b71e765
1192  Author: Teresa Choe <tc2716@columbia.edu>
1193  Date:   Wed Dec 13 16:42:21 2017 -0500
1194
1195      Fixed shift/reduce error
1196
1197  commit 5977da2fd98b9c8243c8ddda4fdeae1641a77c89
1198  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1199  Date:   Wed Dec 13 16:39:29 2017 -0500
1200
1201      more tests
1202
1203  commit e7138eedc37a668fabe528ad1a91c6d393d4898a
1204  Merge: 3bafdd4 81fc78f
1205  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1206  Date:   Wed Dec 13 16:29:13 2017 -0500
1207
```

```
1208      Merge branch 'master' into tests
1209
1210  commit ecd7f34203ed0ee5549df6d1f518cda02bf1fe68
1211  Merge: de0d70b 81fc78f
1212  Author: btsaubt <btsaubt>
1213  Date:   Wed Dec 13 16:28:08 2017 -0500
1214
1215      fixed stupid merge conflictgit status
1216
1217  commit 3bafdd400d2b5785afe32159e2a84b071fb9782e
1218  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1219  Date:   Wed Dec 13 16:24:57 2017 -0500
1220
1221      did stuff
1222
1223  commit de0d70b2316db38cb84ff56756ffbd58d92413eb
1224  Author: btsaubt <btsaubt>
1225  Date:   Wed Dec 13 16:23:43 2017 -0500
1226
1227      commented out some stuff to implement matrices
1228
1229  commit 81fc78f5ee7a1da2a9523391a4822a21159ba186
1230  Merge: 1020b4b e42f2e2
1231  Author: Teresa Choe <tc2716@columbia.edu>
1232  Date:   Wed Dec 13 16:12:37 2017 -0500
1233
1234      All merge conflicts fixed
1235
1236  commit 1020b4bcc6ff826dd35a19851a0f0a9414fe6434
1237  Author: Teresa Choe <tc2716@columbia.edu>
1238  Date:   Wed Dec 13 16:10:05 2017 -0500
1239
1240      operators, fixed tests, added to parser
1241
1242  commit ba09486dd63003077ef02a715db10a27ff41bae3
1243  Author: btsaubt <btsaubt>
1244  Date:   Wed Dec 13 13:09:26 2017 -0500
1245
1246      changes to semant for matrices
1247
1248  commit e42f2e245f32e31905c4a92d60de258f5187fdab
1249  Merge: 539b64c 335840c
1250  Author: btsaubt <btsaubt>
1251  Date:   Tue Dec 12 12:19:02 2017 -0500
1252
1253      Merge branch 'matrices'
1254
1255  commit 335840ce0e42331eece667dfb461f690946d19cf
1256  Author: btsaubt <btsaubt>
1257  Date:   Tue Dec 12 12:18:12 2017 -0500
1258
1259      Merge branch 'master' into matrices
1260
1261  commit c14b5d6c2bddb879c0da046e71b3ee01e33214e8
1262  Author: btsaubt <btsaubt>
1263  Date:   Tue Dec 12 12:16:47 2017 -0500
1264
1265      fixed matrix in grammar
1266
1267  commit 539b64c84b464b6d3c6c002b233e772bb7abb2cc
1268  Merge: 134f8b3 78b698f
1269  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1270  Date:   Tue Dec 12 12:15:55 2017 -0500
```

```
1271
1272      Merge branch 'master' into tests
1273
1274  commit 78b698f7475851b96f21e88fd34f78794f853d9b
1275  Author: btsaubt <btsaubt>
1276  Date:   Tue Dec 12 11:52:12 2017 -0500
1277
1278      added matrices and vectors to grammar (access and initialization)
1279
1280  commit 134f8b37add1b612a409025519ac0609891f77aa
1281  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1282  Date:   Sun Dec 10 20:47:22 2017 -0500
1283
1284      added tests
1285
1286  commit 63239ece5630712731d591f6c8d83fb0fa8a4efc
1287  Author: Teresa Choe <tc2716@columbia.edu>
1288  Date:   Thu Dec 7 19:25:51 2017 -0500
1289
1290      Added types and bitwise to semant.
1291
1292  commit dff6fa813905f48353ead2b80a4c9b7305f97e27
1293  Author: Teresa Choe <tc2716@columbia.edu>
1294  Date:   Thu Dec 7 19:20:41 2017 -0500
1295
1296      Added :) as our single-line comments.
1297
1298  commit 17462a8bff236bfd3a63039f5ac90c84346a836f
1299  Merge: 50082c2 22fc109
1300  Author: Teresa Choe <tc2716@columbia.edu>
1301  Date:   Wed Nov 29 21:02:28 2017 -0500
1302
1303      Finished grammar
1304
1305  commit 50082c2b6ebb954886d1d2ab3c3a0ea7d81029ac
1306  Author: Teresa Choe <tc2716@columbia.edu>
1307  Date:   Wed Nov 29 21:00:56 2017 -0500
1308
1309      Finished most of grammar
1310
1311  commit 22fc1097f7a19456d30d335aca3430972d94d86f
1312  Merge: f959300 1a2af77
1313  Author: btsaubt <bt2420@columbia.edu>
1314  Date:   Wed Nov 29 20:32:32 2017 -0500
1315
1316      Merge branch 'master' into tc2716
1317
1318  commit 1a2af77659e7dac9fb94c5f17ee90968e45bca2d
1319  Merge: 52e678d 683fc9b
1320  Author: Teresa Choe <tc2716@columbia.edu>
1321  Date:   Sun Nov 19 22:55:12 2017 -0500
1322
1323      Merging branches
1324
1325  commit 52e678dc3eac801edcc1a3bd904216e7a85b6ae5
1326  Merge: 09a9d43 1b09c78
1327  Author: Teresa Choe <tc2716@columbia.edu>
1328  Date:   Sun Nov 19 16:57:20 2017 -0500
1329
1330      Changes to scanner.
1331
1332  commit f959300aa3f8ffbc3b0ac100aae3a73773897de2
1333  Merge: 683fc9b 2bb1bb4
```

```
Author: btsaubt <bt2420@columbia.edu>
Date:   Fri Nov 17 22:19:36 2017 -0500

    Merge branch 'tc2716' of https://github.com/btsaubt/pixelman into
        tc2716

    Merge master

commit 2bb1bb435bbcbbc8a802b0e34ff79cab126ac5bb
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Sun Nov 19 22:00:05 2017 -0500

    Added data types and more robust testing for pattern matching.

commit 9777dd890907ccd3af0b64dc914cc35a4e7a9092
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Sun Nov 19 18:36:02 2017 -0500

    Added bitshift and float to ast and parser.

commit 683fc9b17666589d1561797da6698c93d18bc55d
Merge: 09a9d43 ea4fa7a
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Fri Nov 17 14:28:09 2017 -0500

    test 2

commit 09a9d43d4bafba4177327b3d5b54c81d7415a73d
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Fri Nov 17 14:17:22 2017 -0500

    Successful print of hello world and test

commit 9cf711ba994e9f12a7cc8f97f296d9b5bb588a02
Author: btsaubt <bt2420@columbia.edu>
Date:   Fri Nov 17 13:25:47 2017 -0500

    fixed some errors, still error in function checking when it comes to
        return type in line 49

commit b6425b89fe9c727b3f79f9a9fc3c4ba8e3588caa
Author: btsaubt <bt2420@columbia.edu>
Date:   Thu Nov 16 21:55:10 2017 -0500

    fixes to semant and ast, semant still has ocaml syntax errors

commit 6524d2d43f49558671f8ba9ce1e376d6dc63150f
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:   Mon Nov 13 21:39:17 2017 -0500

    hello world

commit ea4fa7a19caa4c997556fec290b667d10dea36dd
Merge: 948d148 1b09c78
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Sun Nov 12 18:38:37 2017 -0500

    Updated scanner

commit 1b09c788c2698787895c5d1963c55cf7ec4693d0
Author: Teresa Choe <tc2716@columbia.edu>
Date:   Sun Nov 12 18:23:24 2017 -0500
```

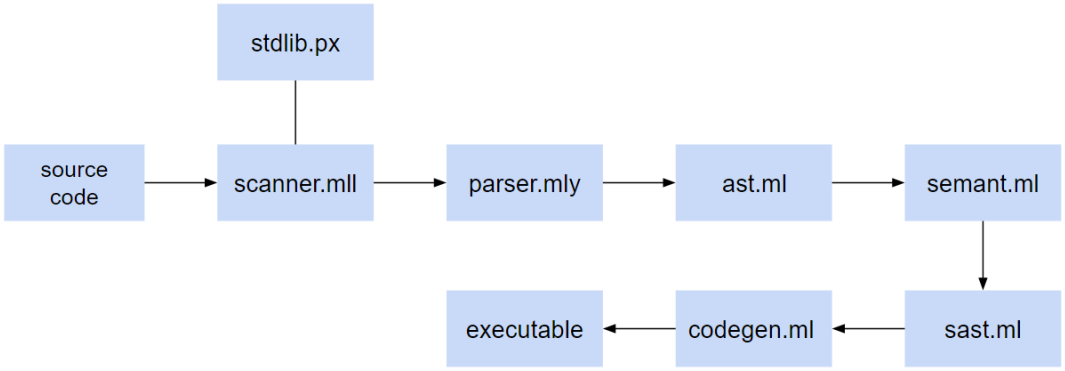
```
1395      Added divint
1396
1397  commit 7e3fe5cef710cb1718d225916173aa178e42efdb
1398  Author: Teresa Choe <tc2716@columbia.edu>
1399  Date:   Sun Nov 12 18:17:42 2017 -0500
1400
1401      Updated scanner to have everything.
1402
1403  commit 948d148a4d5b15633c55e7166ad254df1a87dcfd
1404  Author: btsaubt <bt2420@columbia.edu>
1405  Date:   Sun Nov 12 17:14:26 2017 -0500
1406
1407      removed log
1408
1409  commit 2871c1b53232b5b760d173c7076c35c2297754d5
1410  Author: btsaubt <bt2420@columbia.edu>
1411  Date:   Sun Nov 12 17:12:38 2017 -0500
1412
1413      fixed char and string type
1414
1415  commit 4fabfa54e28ce7352e65bdd2a77aabe569b200da
1416  Merge: cb3d38e 2657a47
1417  Author: Teresa Choe <tc2716@columbia.edu>
1418  Date:   Sun Nov 12 16:45:00 2017 -0500
1419
1420      merging into master
1421
1422  commit cb3d38e3cf9d448ea1a7fb831224b9bbf2adeffa
1423  Author: Teresa Choe <tc2716@columbia.edu>
1424  Date:   Tue Nov 7 14:27:45 2017 -0500
1425
1426      Renamed variables.
1427
1428  commit 2657a4761f69451b64d0872116b75015f9c3d5ed
1429  Author: btsaubt <bt2420@columbia.edu>
1430  Date:   Tue Nov 7 14:23:51 2017 -0500
1431
1432      removed object file
1433
1434  commit 45300b28f403a8dd3ffaa478f5b7293f0d36a5e7
1435  Author: anthony chan <tn.chan5@gmail.com>
1436  Date:   Tue Nov 7 14:20:39 2017 -0500
1437
1438      check for protected functions in semant.ml
1439
1440  commit 64ff70b614c3c69ae6611bccbb7aa6dca89e05db
1441  Author: Teresa Choe <tc2716@columbia.edu>
1442  Date:   Tue Nov 7 11:13:45 2017 -0500
1443
1444      Parser
1445
1446  commit d49db8d1e2991d7db626050cc590bec2b921be25
1447  Author: btsaubt <bt2420@columbia.edu>
1448  Date:   Mon Nov 6 15:42:10 2017 -0500
1449
1450      Added .gitignore file
1451
1452  commit bb1e5b20be2e1b10325e199df6e801bed69358c2
1453  Merge: 2698cf2 4938580
1454  Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1455  Date:   Mon Nov 6 15:36:27 2017 -0500
1456
1457      Merge branch 'master' into ast
```

```
1458
1459 commit 2698cf216cbfe642ddfa4f906f967cac0f0cd0b3
1460 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1461 Date:   Mon Nov 6 15:34:58 2017 -0500
1462
1463     fix Literal
1464
1465 commit 1fa6108d0f978f37a04611757bb43aea43900b46
1466 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1467 Date:   Mon Nov 6 15:33:30 2017 -0500
1468
1469     more changes
1470
1471 commit 4938580bb5fac381578a7985c6d75a72bd9d6b8b
1472 Author: btsaubt <bt2420@columbia.edu>
1473 Date:   Mon Nov 6 15:29:20 2017 -0500
1474
1475     removed float
1476
1477 commit a9eff1c13405c19285662cc14349e9a20d20ace8
1478 Author: btsaubt <bt2420@columbia.edu>
1479 Date:   Mon Nov 6 15:11:50 2017 -0500
1480
1481     added floats, chars, strings, lists to scanner
1482
1483 commit 73b7f9b30dcea28a1cb86935b1f7b99ba72434d4
1484 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1485 Date:   Sun Oct 29 18:04:02 2017 -0400
1486
1487     Added operators to ast
1488
1489 commit bc3cb0f0dae025b2c1bc656aa2fd8ddb657661c0
1490 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1491 Date:   Sun Oct 29 17:48:39 2017 -0400
1492
1493     added our types to ast
1494
1495 commit cb4fb1fd683fcf84742b188dd9f2fa6fb5987983
1496 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1497 Date:   Sun Oct 29 16:56:50 2017 -0400
1498
1499     revert
1500
1501 commit a335920f4532751773f7d13ede9c0150eb0bbe1a
1502 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1503 Date:   Sun Oct 29 16:56:22 2017 -0400
1504
1505     Revert to initial commit
1506
1507 commit 8513b09308e175e5ba47d5bfd375d96bf7f6af69
1508 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1509 Date:   Sun Oct 29 16:47:59 2017 -0400
1510
1511     test
1512
1513 commit a4317c703a7f26bf03ce165b0ed1ad4f415f7f01
1514 Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
1515 Date:   Sun Oct 29 16:44:56 2017 -0400
1516
1517     replace microc
1518
1519 commit ce4ad439ac21b573be3a25abcac5934129e2d3d3
1520 Author: Brian Tsau <bt2420@columbia.edu>
```

```
Date:   Sun Oct 29 14:28:32 2017 -0400

    added micro c files

commit a82ba9cd45dbed385e885e301bdde5078eee9b9e
Author: Brian Tsau <bt2420@columbia.edu>
Date:   Sun Oct 29 14:17:13 2017 -0400

    first commit
```

5 Architectural Design

5.1 Diagram



5.2 Scanner

The scanner, `scanner.mll`, is written in OCamlLex, takes in as input the source code, and tokenizes it. It removes all whitespace, and any text contained within multi-line comments or after a single-line comment token is ignored. If there are characters that cannot be tokenized, the scanner throws an error. The standard library written in pixelman is appended to the source code, and remains a part of the code throughout compilation.

5.3 Parser

The parser, `parser.mly`, is written in OCamlYacc, takes in the stream of tokens and then generates an abstract syntax tree as defined in `ast.ml` using grammar rules in `parser.mly`. The grammar is defined using productions and rules.

5.4 Semantic Checking

The semantic checker, `semant.ml`, is written in Ocaml, and takes in an AST that has been created after parsing, and checks for semantic correctness. This includes type checking for operators, assignment, variable calls, and performs some implicit typecasting for int to float types. Then, it generates a semantically checked AST, defined in `sast.ml`, that contains the inferred types.

5.5 Code Generation

The final component of the compiler, `codegen.ml`, is written in Ocaml. It takes in a semantically checked AST, and attempts to generate LLVM code for each node in the tree. In this part, a C library is linked in to perform some functions needed in our language.

6 Test Plan

6.1 Source Programs

6.1.1 Sample Program 1

Hello World in pixelman

```
def int main()
{
  print_string("Hello World!");
  return 0;
}
```

LLVM Code Generated

```
define i32 @main() {
entry:
  %print_string = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([3 x i8], [3 x i8]* @fmt.170, i32 0, i32 0), i8* getelementptr
     inbounds ([13 x i8], [13 x i8]* @s.173, i32 0, i32 0))
  ret i32 0
}
```

6.1.2 Sample Program 2

Adding two vectors in pixelman

```
def int main(){
    int[3] a;
    int[3] b;
    int i;
    a = [1,2,3];
    b = [1,1,1];
    a = a + b;
    for(i=0;i<3;i=i+1){
        print_int(a[i]);
    }
    return 0;
}
```

LLVM Code Generated

```
define i32 @main() {
entry:
  %a = alloca [3 x i32]
  %b = alloca [3 x i32]
  %i = alloca i32
  store [3 x i32] [i32 1, i32 2, i32 3], [3 x i32]* %a
  store [3 x i32] [i32 1, i32 1, i32 1], [3 x i32]* %b
  %b1 = load [3 x i32], [3 x i32]* %b
  %a2 = load [3 x i32], [3 x i32]* %a
  %vec_vec_addi_result = call [3 x i32] @vec_vec_addi([3 x i32] %a2, [3
      x i32] %b1)
  store [3 x i32] %vec_vec_addi_result, [3 x i32]* %a
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body,
     %entry
  %i7 = load i32, i32* %i
  %tmp8 = icmp slt i32 %i7, 3
```

```
18    br i1 %tmp8, label %while_body, label %merge
19
20  while_body:                                       ; preds = %while
21    %i3 = load i32, i32* %i
22    %a4 = getelementptr [3 x i32], [3 x i32]* %a, i32 0, i32 %i3
23    %a5 = load i32, i32* %a4
24    %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
          i8], [3 x i8]* @fmt.169, i32 0, i32 0), i32 %a5)
25    %i6 = load i32, i32* %i
26    %tmp = add i32 %i6, 1
27    store i32 %tmp, i32* %i
28    br label %while
29
30  merge:                                            ; preds = %while
31    ret i32 0
32  }
```

6.1.3 Sample Program 3

Reversing a vector in pixelman

```
1  def int main()
2  {
3      int[10] a;
4      int start;
5      int end;
6      int temp;
7      a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
8      start = 0;
9      end = sizeof(a) - 1;
10     while (start < end && start!= end){
11         temp = a[start];
12         a[start] = a[end];
13         a[end] = temp;
14         start = start+1;
15         end = end-1;
16     }
17     return 0;
18 }
```

LLVM Code Generated

```
1  define i32 @main() {
2  entry:
3    %a = alloca [10 x i32]
4    %start = alloca i32
5    %end = alloca i32
6    %temp = alloca i32
7    store [10 x i32] [i32 1, i32 2, i32 3, i32 4, i32 5, i32 6, i32 7, i32
         8, i32 9, i32 10], [10 x i32]* %a
8    store i32 0, i32* %start
9    store i32 9, i32* %end
10   br label %while
11
12 while:                                            ; preds = %while_body,
       %entry
13   %start15 = load i32, i32* %start
14   %end16 = load i32, i32* %end
15   %tmp17 = icmp slt i32 %start15, %end16
16   %start18 = load i32, i32* %start
17   %end19 = load i32, i32* %end
18   %tmp20 = icmp ne i32 %start18, %end19
19   %tmp21 = and i1 %tmp17, %tmp20
```

```
20    br i1 %tmp21, label %while_body, label %merge
21
22  while_body:                                       ; preds = %while
23    %start1 = load i32, i32* %start
24    %a2 = getelementptr [10 x i32], [10 x i32]* %a, i32 0, i32 %start1
25    %a3 = load i32, i32* %a2
26    store i32 %a3, i32* %temp
27    %start4 = load i32, i32* %start
28    %a5 = getelementptr [10 x i32], [10 x i32]* %a, i32 0, i32 %start4
29    %end6 = load i32, i32* %end
30    %a7 = getelementptr [10 x i32], [10 x i32]* %a, i32 0, i32 %end6
31    %a8 = load i32, i32* %a7
32    store i32 %a8, i32* %a5
33    %end9 = load i32, i32* %end
34    %a10 = getelementptr [10 x i32], [10 x i32]* %a, i32 0, i32 %end9
35    %temp11 = load i32, i32* %temp
36    store i32 %temp11, i32* %a10
37    %start12 = load i32, i32* %start
38    %tmp = add i32 %start12, 1
39    store i32 %tmp, i32* %start
40    %end13 = load i32, i32* %end
41    %tmp14 = sub i32 %end13, 1
42    store i32 %tmp14, i32* %end
43    br label %while
44
45  merge:                                            ; preds = %while
46    ret i32 0
47  }
```

6.2 Test Suite

6.2.1 Test Suite Code

Below is the script that ran all of our tests. All tests were written in files with a .px extension. Tests to pass had the name format "test-test_name.px" and tests to fail had the name format "fail-test_name.px." Each test was matched to an output file and if the output of the test matched the output file, the test would pass. Tests to pass had output files with the name format "test-test_name.out" and tests to fail had the name format "fail-test_name.err." Note that instead of having output, the tests that were written to fail threw an error and the test suite determined if it threw the correct error by matching it to the corresponding .err file.

testall.sh

```
1  #!/bin/sh
2
3  # Regression testing script for MicroC
4  # Step through a list of files
5  #  Compile, run, and check the output of each expected-to-work test
6  #  Compile and check the error of each expected-to-fail test
7
8  # Path to the LLVM interpreter
9  LLI="lli"
10 #LLI="/usr/local/opt/llvm/bin/lli"
11
12 # Path to the LLVM compiler
13 LLC="llc"
14
15 # Path to the C compiler
16 CC="cc"
17
18 # Path to the microc compiler.  Usually "./microc.native"
19 # Try "_build/microc.native" if ocamlbuild was unable to create a
       symbolic link.
20 MICROC="./pixelman.native"
```

```
21 #MICROC="_build/microc.native"
22
23 # Set time limit for all operations
24 ulimit -t 30
25
26 globallog=testall.log
27 rm -f $globallog
28 error=0
29 globalerror=0
30
31 keep=0
32
33 Usage() {
34     echo "Usage: testall.sh [options] [.px files]"
35     echo "-k    Keep intermediate files"
36     echo "-h    Print this help"
37     exit 1
38 }
39
40 SignalError() {
41     if [ $error -eq 0 ] ; then
42         echo "FAILED"
43         error=1
44     fi
45     echo "  $1"
46 }
47
48 # Compare <outfile> <reffile> <difffile>
49 # Compares the outfile with reffile.  Differences, if any, written to
      difffile
50 Compare() {
51     generatedfiles="$generatedfiles $3"
52     echo diff -b $1 $2 ">" $3 1>&2
53     diff -b "$1" "$2" > "$3" 2>&1 || {
54         SignalError "$1 differs"
55         echo "FAILED $1 differs from $2" 1>&2
56     }
57 }
58
59 # Run <args>
60 # Report the command, run it, and report any errors
61 Run() {
62     echo $* 1>&2
63     eval $* || {
64         SignalError "$1 failed on $*"
65         return 1
66     }
67 }
68
69 # RunFail <args>
70 # Report the command, run it, and expect an error
71 RunFail() {
72     echo $* 1>&2
73     eval $* && {
74         SignalError "failed: $* did not report an error"
75         return 1
76     }
77     return 0
78 }
79
80 Check() {
81     error=0
82     basename=`echo $1 | sed 's/.*\\///
```

```
83                                  s/.px//'`
84      reffile=`echo $1 | sed 's/.px$//'`
85      basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
86
87      echo -n "$basename..."
88
89      echo 1>&2
90      echo "###### Testing $basename" 1>&2
91
92      generatedfiles=""
93
94      generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${
            basename}.exe ${basename}.out" &&
95      Run "$MICROC" "<" $1 ">" "${basename}.ll" &&
96      Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
97      Run "$CC" "-o" "${basename}.exe" -nopie "${basename}.s" "printbig.o"
            "makePic.o" "inputPic.o" &&
98      Run "./${basename}.exe" > "${basename}.out" &&
99      Compare ${basename}.out ${reffile}.out ${basename}.diff
100
101     # Report the status and clean up the generated files
102
103     if [ $error -eq 0 ] ; then
104         if [ $keep -eq 0 ] ; then
105             rm -f $generatedfiles
106         fi
107         echo "OK"
108         echo "###### SUCCESS" 1>&2
109     else
110         echo "###### FAILED" 1>&2
111         globalerror=$error
112     fi
113 }
114
115 CheckFail() {
116     error=0
117     basename=`echo $1 | sed 's/.*\///
118                                 s/.px//'`
119     reffile=`echo $1 | sed 's/.px$//'`
120     basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
121
122     echo -n "$basename..."
123
124     echo 1>&2
125     echo "###### Testing $basename" 1>&2
126
127     generatedfiles=""
128
129     generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
130     RunFail "$MICROC" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
131     Compare ${basename}.err ${reffile}.err ${basename}.diff
132
133     # Report the status and clean up the generated files
134
135     if [ $error -eq 0 ] ; then
136         if [ $keep -eq 0 ] ; then
137             rm -f $generatedfiles
138         fi
139         echo "OK"
140         echo "###### SUCCESS" 1>&2
141     else
142         echo "###### FAILED" 1>&2
143         globalerror=$error
```

```
144     fi
145 }
146
147 while getopts kdpsh c; do
148     case $c in
149         k) # Keep intermediate files
150             keep=1
151             ;;
152         h) # Help
153             Usage
154             ;;
155     esac
156 done
157
158 shift `expr $OPTIND - 1`
159
160 LLIFail() {
161   echo "Could not find the LLVM interpreter \"$LLI\"."
162   echo "Check your LLVM installation and/or modify the LLI variable in
        testall.sh"
163   exit 1
164 }
165
166 which "$LLI" >> $globallog || LLIFail
167
168 if [ ! -f printbig.o ]
169 then
170     echo "Could not find printbig.o"
171     echo "Try \"make printbig.o\""
172     exit 1
173 fi
174
175 if [ $# -ge 1 ]
176 then
177     files=$@
178 else
179     files="tests/test-*.px tests/fail-*.px"
180 fi
181
182 for file in $files
183 do
184     case $file in
185         *test-*)
186             Check $file 2>> $globallog
187             ;;
188         *fail-*)
189             CheckFail $file 2>> $globallog
190             ;;
191         *)
192             echo "unknown file type $file"
193             globalerror=1
194             ;;
195     esac
196 done
197
198 exit $globalerror
```

6.2.2 Output of running test suite

The output from running our test suite can be seen below.

```
1 $ ./testall.sh
2 -n test-add1...
3 OK
```

```
 4 -n test-arith1...
 5 OK
 6 -n test-arith2...
 7 OK
 8 -n test-arith3...
 9 OK
10 -n test-ast-arrays...
11 OK
12 -n test-astdatatypes...
13 OK
14 -n test-astfloat...
15 OK
16 -n test-astop...
17 OK
18 -n test-castfloattoint...
19 OK
20 -n test-castinttofloat...
21 OK
22 -n test-codegen-matrixliteral...
23 OK
24 -n test-codegen-vector...
25 OK
26 -n test-codegenoperators...
27 OK
28 -n test-fib...
29 OK
30 -n test-floatopsnospace...
31 OK
32 -n test-for1...
33 OK
34 -n test-for2...
35 OK
36 -n test-func1...
37 OK
38 -n test-func2...
39 OK
40 -n test-func3...
41 OK
42 -n test-func4...
43 OK
44 -n test-func5...
45 OK
46 -n test-func6...
47 OK
48 -n test-func7...
49 OK
50 -n test-func8...
51 OK
52 -n test-gcd...
53 OK
54 -n test-gcd2...
55 OK
56 -n test-global1...
57 OK
58 -n test-global2...
59 OK
60 -n test-global3...
61 OK
62 -n test-hello...
63 OK
64 -n test-if1...
65 OK
66 -n test-if2...
```

```
67  OK
68  -n test-if3 ...
69  OK
70  -n test-if4 ...
71  OK
72  -n test-if5 ...
73  OK
74  -n test-image ...
75  OK
76  -n test-intopsnospace ...
77  OK
78  -n test-local1 ...
79  OK
80  -n test-local2 ...
81  OK
82  -n test-matrixfloat ...
83  OK
84  -n test-matrixfloatdeclare ...
85  OK
86  -n test-matrixint ...
87  OK
88  -n test-matrixintdeclare ...
89  OK
90  -n test-multi-comments ...
91  OK
92  -n test-ops1 ...
93  OK
94  -n test-ops2 ...
95  OK
96  -n test-printb ...
97  OK
98  -n test-printbig ...
99  OK
100 -n test-printfloat ...
101 OK
102 -n test-printstring ...
103 OK
104 -n test-semantdatatypes ...
105 OK
106 -n test-stdlib-casts ...
107 OK
108 -n test-var1 ...
109 OK
110 -n test-var2 ...
111 OK
112 -n test-vector-access-assign ...
113 OK
114 -n test-vector-access ...
115 OK
116 -n test-vector-addition ...
117 OK
118 -n test-vectorfloat ...
119 OK
120 -n test-vectorfloatdeclare ...
121 OK
122 -n test-vectorint ...
123 OK
124 -n test-vectorintdeclare ...
125 OK
126 -n test-while1 ...
127 OK
128 -n test-while2 ...
129 OK
```

```
130 -n fail-assign1...
131 OK
132 -n fail-assign2...
133 OK
134 -n fail-assign3...
135 OK
136 -n fail-dead1...
137 OK
138 -n fail-dead2...
139 OK
140 -n fail-expr1...
141 OK
142 -n fail-expr2...
143 OK
144 -n fail-for1...
145 OK
146 -n fail-for2...
147 OK
148 -n fail-for3...
149 OK
150 -n fail-for4...
151 OK
152 -n fail-for5...
153 OK
154 -n fail-func1...
155 OK
156 -n fail-func10...
157 OK
158 -n fail-func2...
159 OK
160 -n fail-func3...
161 OK
162 -n fail-func4...
163 OK
164 -n fail-func5...
165 OK
166 -n fail-func6...
167 OK
168 -n fail-func7...
169 OK
170 -n fail-func8...
171 OK
172 -n fail-func9...
173 OK
174 -n fail-global1...
175 OK
176 -n fail-global2...
177 OK
178 -n fail-if1...
179 OK
180 -n fail-if2...
181 OK
182 -n fail-if3...
183 OK
184 -n fail-nomain...
185 OK
186 -n fail-return1...
187 OK
188 -n fail-return2...
189 OK
190 -n fail-vectorint...
191 OK
192 -n fail-while1...
```

```
193 OK
194 -n fail-while2...
195 OK
```

6.2.3 How Test Cases Were Chosen

We chose to create unit tests for all of the features we implemented in our language. We chose to use unit tests to ensure that every part of our language worked correctly. We also chose to use unit tests so that we could easily pinpoint where any errors were if we encountered them.

6.2.4 Automation Used While Testing

We used a script called testall.sh to automate our test suite. It ran all of our tests and displayed whether each test passed or failed. It easily ran in approximately 2 seconds and could be ran by calling ./testall.sh.

6.3 Responsibilities

Everybody wrote tests after implementing any new features to confirm that they worked as expected. Gabe created the test suite and added everyone's new tests to the test suite when a new feature was implemented.

7 Lessons Learned

Anthony

Learning two new languages (Ocaml and LLVM), functional programming, and compiler design is difficult. Balance the workload better from other courses in the semester and don't overload too much. Start early on the project and make progress early and consistently. Everything is interconnected and understanding things early on makes it easier to work on the project later. Also, make more use of the TAs for help with things that are not working or things that you don't have a good grasp of.

Brian

This was the first semester-long programming group project that I have completed at Columbia - it takes a LOT of work. Start early. Don't stop after you reach a certain milestone early on - there is a pretty good chance that you will have to change a design decision later on when you try to implement another feature. We gave ourselves too little time to complete our project, so we weren't able to implement everything we wanted to, and had to water down our language from the LRM. The structure of compilers and LLVM generation has a pretty steep learning curve, so allocating enough time early on to understand everything in MicroC is essential. It will be pretty intimidating at first, but the more time you spend chipping away at it, the easier it becomes to implement your own compiler.

Gabe

Don't just start early, do as much work as you can early so you have time at the end of the semester to add cool features to your language. Once you have a working language, that is still only the beginning. If you really want to make a great language, make sure you have a working language well before reading week so you can spend time making your language do exactly what you want it to do while everyone else is just still just trying to create a working language. Also, communication is extremely important when working on a common code base with a group. If everyone in the group knows what everyone else is working on, you will have a lot less trouble with merge conflicts and breaking the program.

Teresa

START EARLY! Also, it is not enough to write code as if in a bubble, but to be cognizant of the way the code I write will affect other files on other branches. It is also really important to be careful maintaining the code, because it gets really hard to debug. All the things I learned from professional coders in my work experience seemed so relevant and key to coding efficiently. It is not enough that we learn; in fact, to truly understand it took me full commitment to breaking code and making

mistakes! Even starting on something really small is a contribution to the code that is ours. So it was really cool realizing how impactful coding can be.

8 Appendix

8.1 ast.ml

```
(* Abstract Syntax Tree and functions for printing it
 * Anthony Chan
 * Gabriel Kramer-Garcia
 * Brian Tsau
 * Teresa Choe
*)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Shiftleft | Shiftright | Bitand | Bitor | Bitxor | Mod

type uop = Neg | Not | IntCast | FloatCast

type expr =
    Int_Literal of int
  | Float_Literal of float
  | Char_Literal of char
  | String_Literal of string
  | Vector_Literal of expr list
  | Matrix_Literal of expr list list
  | Bool_Literal of bool
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of string * expr list
  | VecAccess of string * expr
  | MatAccess of string * expr * expr
  | MatRow of string * expr
  | MatCol of string * expr
  | SizeOf of string
  (*| ImAccess of string * int*)
  | Noexpr

type typ =
    Int
  | Bool
  | Float
  | Char
  | String
  | Void
  (*| Image of expr * expr *)
  | Vector of typ * expr
  | Matrix of typ * expr * expr

type bind = typ * string

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
```

```
54    (* | Break
55    | Continue  *)
56
57  type func_decl = {
58      typ : typ;
59      fname : string;
60      formals : bind list;
61      locals : bind list;
62      body : stmt list;
63    }
64
65  type program = bind list * func_decl list
66
67  (* Pretty-printing functions *)
68
69  let string_of_op = function
70      Add -> "+"
71    | Sub -> "-"
72    | Mult -> "*"
73    | Div -> "/"
74    | Equal -> "=="
75    | Neq -> "!="
76    | Less -> "<"
77    | Leq -> "<="
78    | Greater -> ">"
79    | Geq -> ">="
80    | And -> "&&"
81    | Or -> "||"
82    | Mod -> "%"
83    | Shiftleft -> "<<"
84    | Shiftright -> ">>"
85    | Bitand -> "&"
86    | Bitor -> "|"
87    | Bitxor -> "^"
88
89  let string_of_uop = function
90      Neg -> "-"
91    | Not -> "!"
92    | IntCast -> "(Int) "
93    | FloatCast -> "(Float) "
94
95  let rec string_of_vector el =
96    "[" ^ String.concat ", " (List.map (fun e -> string_of_expr e) el) ^
        "]"
97
98  and(* rec *) string_of_matrix el = "[|" ^
99     String.concat " & " (List.map (fun v -> string_of_vector v) el) ^
         "|]"
100
101 and(* rec *) string_of_expr = function
102     Int_Literal(i) -> string_of_int i
103   | Float_Literal(f) -> string_of_float f
104   | Char_Literal(c) -> Char.escaped c
105   | String_Literal(s) -> s
106   | Bool_Literal(b) -> if b then "true" else "false"
107   | Vector_Literal(el) -> string_of_vector el
108   | Matrix_Literal(el) -> string_of_matrix el
109   | Id(s) -> s
110   | Binop(e1, o, e2) ->
111       string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
112   | Unop(o, e) -> string_of_uop o ^ string_of_expr e
113   | Assign(e1, e2) -> string_of_expr e1  ^ " = " ^ string_of_expr e2
114 (*    | Assign(v, e2) -> v ^ " = " ^ string_of_expr e2 *)
```

```
115   | SizeOf(s) -> "sizeof(" ^ s ^ ")"
116   | Call(f, el) ->
117       f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
118   | VecAccess(v, e) -> v ^ "[" ^ string_of_expr e ^ "]"
119   | MatAccess(v, e1, e2) -> v ^ "[" ^ string_of_expr e1 ^ "]" ^
120                             "[" ^ string_of_expr e2 ^ "]"
121   | MatRow(v, e) -> v ^ "[" ^ string_of_expr e ^ "][]"
122   | MatCol(v, e) -> v ^ "[][" ^ string_of_expr e ^ "]"
123   (*| ImAccess(v, c) -> v ^ ".[" ^ string_of_int c ^ "]"*)
124   | Noexpr -> ""
125
126 let rec string_of_stmt = function
127     Block(stmts) ->
128       "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
129   | Expr(expr) -> string_of_expr expr ^ ";\n";
130   | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
131   | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
        string_of_stmt s
132   | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
133       string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
134   | For(e1, e2, e3, s) ->
135       "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
136       string_of_expr e3  ^ ") " ^ string_of_stmt s
137   | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt
        s
138 (*    | Break -> "break;"
139   | Continue -> "continue;" *)
140
141 let rec string_of_typ = function
142     Int -> "int"
143   | Bool -> "bool"
144   | Char -> "char"
145   | Float -> "float"
146   | String -> "string"
147   | Void -> "void"
148   (*| Image(h, w) -> "Image[" ^ string_of_expr h ^ "," ^ string_of_expr
        w ^ "]"*)
149   | Vector(t, e) -> string_of_typ t ^ "[" ^ string_of_expr e ^ "]"
150   | Matrix(t, e1, e2) -> string_of_typ t ^ "[" ^ string_of_expr e1 ^
        "][" ^ string_of_expr e2 ^ "]"
151
152 let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"
153
154 let string_of_fdecl fdecl =
155   "def " ^ string_of_typ fdecl.typ ^ " " ^
156   fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
157   ")\n{\n" ^
158   String.concat "" (List.map string_of_vdecl fdecl.locals) ^
159   String.concat "" (List.map string_of_stmt fdecl.body) ^
160   "}\n"
161
162 let string_of_program (vars, funcs) =
163   String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
164   String.concat "\n" (List.map string_of_fdecl funcs)
```

8.2 codegen.ml

```
1  (* pixelman's Code generation: translate takes a semantically checked
       AST and produces LLVM IR
2   * http://llvm.org/docs/tutorial/index.html
3   * http://llvm.moe/
4   * http://llvm.moe/ocaml/
5   * Teresa Choe
6   * Brian Tsau
7   * Anthony Chan
8   * Gabriel Kramer-Garcia
9  *)
10
11 module L = Llvm
12 module A = Ast
13 module S = Sast
14
15 module StringMap = Map.Make(String)
16
17 let translate (globals, functions) =
18   let context = L.global_context () in
19   let the_module = L.create_module context "Pixelman"
20   and i32_t    = L.i32_type   context
21   and i8_t     = L.i8_type    context
22   and i1_t     = L.i1_type    context
23   and f_t      = L.double_type context
24   and array_t  = L.array_type
25   and void_t   = L.void_type  context in
26
27   let int_lit_to_int = function
28     A.Int_Literal(i) -> i | _ -> raise(Failure("Can only make vector/
         matrix of dimension int literal"))
29   in
30   let ltype_of_typ = function
31       A.Int -> i32_t
32     | A.Float -> f_t
33     | A.Bool -> i1_t
34     | A.Char -> i8_t
35     | A.String -> i32_t
36     | A.Void -> void_t
37     | A.Vector(typ, size) -> (match typ with
38                                A.Int -> array_t i32_t (int_lit_to_int size
                                     )
39                              | A.Float -> array_t f_t (int_lit_to_int
                                   size)
40                              | _ -> raise(Failure("Can only make vector
                                   of type int/float")))
41     | A.Matrix(t, s1, s2) -> (match t with
42                             A.Int -> array_t (array_t i32_t (
                                  int_lit_to_int s2)) (int_lit_to_int s1)
43                              | A.Float -> array_t (array_t f_t (
                                   int_lit_to_int s2)) (int_lit_to_int s1)
44                              | _ -> raise(Failure("Cannot only make
                                   vector of type int/float")))
45     (*| A.Image(h,w) -> let mat_t = ltype_of_typ (A.Matrix(A.Int, h, w))
46                        in array_t mat_t 3 (* make an array of 3 h x w
                              matrices *)*)
47   in
48   (* Declare each global variable; remember its value in a map *)
49   let global_vars =
50     let global_var m (t, n) =
51             let init = if ltype_of_typ t != f_t then
52                          L.const_int (ltype_of_typ t) 0
```

```
53                          else
54                              L.const_float (ltype_of_typ t) 0.0
55        in StringMap.add n (L.define_global n init the_module) m in
56      List.fold_left global_var StringMap.empty globals in
57
58    (* Declare printf(), which the print built-in function will call *)
59    let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |]
          in
60    let printf_func = L.declare_function "printf" printf_t the_module in
61
62    (* Declare the built-in printbig() function *)
63    let printbig_t = L.function_type i32_t [| i32_t |] in
64    let printbig_func = L.declare_function "printbig" printbig_t
          the_module in
65
66    let makePic_t = L.function_type i32_t [| i32_t; i32_t; i32_t; i32_t;
          i32_t |] in
67    let makePic_func = L.declare_function "makePic" makePic_t the_module
          in
68
69    let inputPic_t = L.function_type i32_t [| i32_t |] in
70    let inputPic_func = L.declare_function "inputPic" inputPic_t
          the_module in
71
72    (* Define each function (arguments and return type) so we can call it
          *)
73    let function_decls =
74      let function_decl m fdecl =
75        let name = fdecl.S.sfname
76        and formal_types =
77                Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
                    fdecl.S.sformals)
78        in let ftype = L.function_type (ltype_of_typ fdecl.S.styp)
              formal_types in
79        StringMap.add name (L.define_function name ftype the_module, fdecl
              ) m in
80      List.fold_left function_decl StringMap.empty functions in
81
82    (* Fill in the body of the given function *)
83    let build_function_body fdecl =
84      let (the_function, _) = StringMap.find fdecl.S.sfname function_decls
            in
85      let builder = L.builder_at_end context (L.entry_block the_function)
          in
86
87      let int_format_str = L.build_global_stringptr "%d" "fmt" builder in
88      let string_format_str = L.build_global_stringptr "%s" "fmt" builder
          in
89      let nl_format_str = L.build_global_stringptr "\n" "fmt" builder in
90      let float_format_str = L.build_global_stringptr "%f" "fmt" builder
          in
91
92      (* Construct the function's "locals": formal arguments and locally
93         declared variables.  Allocate each on the stack, initialize their
94         value, if appropriate, and remember their values in the "locals"
               map *)
95      let local_vars =
96        let add_formal m (t, n) p = L.set_value_name n p;
97          let local = L.build_alloca (ltype_of_typ t) n builder in
98          ignore (L.build_store p local builder);
99          StringMap.add n local m
100       in
101
```

```
102       let add_local m (t, n) =
103             let local_var = L.build_alloca (ltype_of_typ t) n builder
104             in StringMap.add n local_var m
105       in
106       let formals = List.fold_left2 add_formal StringMap.empty fdecl.S.
              sformals
107             (Array.to_list (L.params the_function))
108       in
109       List.fold_left add_local formals fdecl.S.slocals
110     in
111
112     (* Return the value for a variable or formal argument *)
113     let lookup n = try StringMap.find n local_vars
114                    with Not_found -> StringMap.find n global_vars
115     in
116
117     let rec get_vector_acc_addr s e1 builder = L.build_gep (lookup s)
118         [| (L.const_int i32_t 0); (expr builder e1) |] s builder
119
120     and get_matrix_acc_addr s e1 e2 builder = L.build_gep (lookup s)
121         [| L.const_int i32_t 0; expr builder e1; expr builder e2 |] s
              builder
122
123     (* Construct code for an expression; return its value *)
124     and expr builder = function
125           S.SInt_Literal i -> L.const_int i32_t i
126       | S.SFloat_Literal fl -> L.const_float f_t fl
127       | S.SChar_Literal c -> L.const_int i8_t (Char.code c)
128       | S.SString_Literal s -> L.build_global_stringptr s "s" builder
129       | S.SBool_Literal b -> L.const_int i1_t (if b then 1 else 0)
130       | S.SVector_Literal (l, t) -> L.const_array (ltype_of_typ t) (
            Array.of_list (List.map (expr builder) l))
131       | S.SMatrix_Literal (ell, t) -> (match t with
132           A.Matrix(A.Float,_,_) ->
133             let order = List.map List.rev ell in
134             let f_lists = List.map (List.map (expr builder)) order in
135             let array_list = List.map Array.of_list f_lists in
136             let f_list_list = List.map (L.const_array f_t) array_list
                  in
137             let array_of_arrays = Array.of_list f_list_list in
138               L.const_array(array_t f_t (List.length (List.hd ell)))
                    array_of_arrays
139         | A.Matrix(A.Int,_,_) ->
140             let order = List.map List.rev ell in
141             let i_lists = List.map (List.map (expr builder)) order in
142             let array_list = List.map Array.of_list i_lists in
143             let i_list_array = List.map (L.const_array i32_t) array_list
                  in
144             let array_of_arrays = Array.of_list i_list_array in
145               L.const_array(array_t i32_t (List.length (List.hd ell)))
                    array_of_arrays
146         | _ -> raise(Failure(A.string_of_typ t))
147       )
148       | S.SNoexpr -> L.const_int i32_t 0
149       | S.SId (s, _) -> L.build_load (lookup s) s builder
150       | S.SSizeOf(vm,_) -> L.const_int i32_t (L.array_length (L.
            element_type (L.type_of (lookup vm))))
151       | S.SVecAccess(s, e, _) -> L.build_load (get_vector_acc_addr s e
            builder) s builder
152       | S.SMatAccess(s, e1, e2, _) -> L.build_load (get_matrix_acc_addr
            s e1 e2 builder) s builder
153       | S.SBinop (e1, op, e2, _) -> (* too late to implement using sexpr
             types to make things easier *)
```

```
154             let e1' = expr builder e1
155             and e2' = expr builder e2 in
156             (match op with
157             (*  A.Add     -> L.build_add *)
158               A.Add -> (let e1_type_string = L.string_of_lltype (L.type_of
                      e1') in
159                         (match e1_type_string with
160                            "double" -> L.build_fadd
161                          | "i32" -> L.build_add
162                          | _ -> raise(Failure("Illegal type operation")) )
                            )
163             | A.Sub      -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
164                             (match e1_type_string with
165                               "double" -> L.build_fsub
166                             | "i32"  -> L.build_sub
167                             | _ -> raise(Failure("Illegal type operation")
                                ) ))
168             | A.Mod      -> L.build_urem
169             | A.Mult     -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
170                             (match e1_type_string with
171                             "double" -> L.build_fmul
172                              | "i32"  -> L.build_mul
173                              | _ -> raise(Failure("illegal type operation
                                ")) ))
174             | A.Div      -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
175                             (match e1_type_string with
176                                "double" -> L.build_fdiv
177                              | "i32"  -> L.build_sdiv
178                              | _ -> raise(Failure("illegal type operation
                                ")) ))
179             | A.And      -> L.build_and
180                 | A.Or       -> L.build_or
181             | A.Bitxor   -> L.build_xor
182             | A.Bitand   -> L.build_and
183             | A.Bitor    -> L.build_or
184             | A.Shiftright -> L.build_lshr
185             | A.Shiftleft -> L.build_shl
186             | A.Equal    -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
187                             (match e1_type_string with
188                             "double" -> L.build_fcmp L.Fcmp.Oeq
189                              | "i32" -> L.build_icmp L.Icmp.Eq
190                              | _ -> raise(Failure("Illegal type operation
                                ")) ))
191             | A.Neq      -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
192                             (match e1_type_string with
193                             "double" -> L.build_fcmp L.Fcmp.One
194                              | "i32" -> L.build_icmp L.Icmp.Ne
195                              | _ -> raise(Failure("Illegal type operation
                                ")) ))
196             | A.Less     -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
197                             (match e1_type_string with
198                             "double" -> L.build_fcmp L.Fcmp.Olt
199                              | "i32" -> L.build_icmp L.Icmp.Slt
200                              | _ -> raise(Failure("Illegal type operation
                                ")) ))
201             | A.Leq      -> (let e1_type_string = L.string_of_lltype (L.
                  type_of e1') in
```

```
202                         (match e1_type_string with
203                         "double" -> L.build_fcmp L.Fcmp.Ole
204                          | "i32"  -> L.build_icmp L.Icmp.Sle
205                          | _ -> raise(Failure("Illegal type operation
                                "))  ))
206           | A.Greater -> (let e1_type_string = L.string_of_lltype (L.
                type_of e1') in
207                         (match e1_type_string with
208                         "double" -> L.build_fcmp L.Fcmp.Ogt
209                          | "i32"  -> L.build_icmp L.Icmp.Sgt
210                          | _ -> raise(Failure("Illegal type operation"
                                ))  ))
211           | A.Geq    -> (let e1_type_string = L.string_of_lltype (L.
                type_of e1') in
212                         (match e1_type_string with
213                         "double" -> L.build_fcmp L.Fcmp.Oge
214                          | "i32"  -> L.build_icmp L.Icmp.Sge
215                          | _ -> raise(Failure("Illegal type operation"
                                ))  ))
216           ) e1' e2' "tmp" builder
217       | S.SUnop(op, e, t) -> let e' = expr builder e in
218             (match op with
219                 A.Neg         -> (if t == A.Float then L.build_fneg else
                        L.build_neg) e' "tmp" builder
220           | A.Not       -> L.build_not e' "tmp" builder
221           | A.IntCast   -> L.build_fptosi e' i32_t "float_to_int"
                builder
222           | A.FloatCast -> L.build_sitofp e' f_t   "int_to_float"
                builder )
223       | S.SAssign (v, e, _) -> let lsb = (match v with
224                                   S.SId(n,_) -> lookup n
225                                   | S.SVecAccess(s,e,_) ->
                                      get_vector_acc_addr s e builder
226                                   | S.SMatAccess(s,e1,e2,_) ->
                                      get_matrix_acc_addr s e1 e2 builder
227                                   | _ -> raise(Failure("Illegal
                                      assignment lvalue")))
228                                in
229                                let rsb = expr builder e in
230                                ignore (L.build_store rsb lsb builder);
                                   rsb
231       | S.SCall ("print_int", [e], _) ->
232                L.build_call printf_func [| int_format_str ; (expr
                    builder e) |]
233                "printf" builder
234       | S.SCall ("print_string", [e], _) ->
235                L.build_call printf_func [| string_format_str ; (expr
                    builder e) |]
236                "print_string" builder
237       | S.SCall ("print_newline", [], _) ->
238                L.build_call printf_func [| nl_format_str |]
239                "print_newline" builder
240       | S.SCall ("print_float", [e], _) ->
241                L.build_call printf_func [| float_format_str ; (expr
                    builder e) |]
242                "print_float" builder
243       | S.SCall ("printbig", [e], _) ->
244                L.build_call printbig_func [| (expr builder e) |] "
                    printbig" builder
245       | S.SCall ("makePic", [e;e1;e2;e3;e4], _) ->
246             L.build_call makePic_func [| (expr builder e);(expr builder e1
                );(expr builder e2);(expr builder e3);(expr builder e4) |]
                 "makePic" builder
```

```
247       | S.SCall ("inputPic", [e], _) ->
248           L.build_call inputPic_func [| (expr builder e) |] "inputPic"
                builder
249       | S.SCall (f, act, _) ->
250         let (fdef, fdecl) = StringMap.find f function_decls in
251           let actuals = List.rev (List.map (expr builder) (List.rev act
                )) in
252           let result = (match fdecl.S.styp with A.Void -> ""
253                                               | _      -> f ^ "_result"
                                                    ) in
254           L.build_call fdef (Array.of_list actuals) result builder
255         in
256     (* Invoke "f builder" if the current block doesn't already
257        have a terminal (e.g., a branch). *)
258     let add_terminal builder f =
259       match L.block_terminator (L.insertion_block builder) with
260         Some _ -> ()
261       | None -> ignore (f builder) in
262
263     (* Build the code for the given statement; return the builder for
264        the statement's successor *)
265     let rec stmt builder = function
266         S.SBlock sl -> List.fold_left stmt builder sl
267       | S.SExpr e -> ignore (expr builder e); builder
268       | S.SReturn e -> ignore (match fdecl.S.styp with
269           A.Void -> L.build_ret_void builder
270         | _ -> L.build_ret (expr builder e) builder); builder
271       | S.SIf (predicate, then_stmt, else_stmt) ->
272         let bool_val = expr builder predicate in
273         let merge_bb = L.append_block context "merge" the_function in
274
275         let then_bb = L.append_block context "then" the_function in
276         add_terminal (stmt (L.builder_at_end context then_bb) then_stmt
                )
277           (L.build_br merge_bb);
278
279         let else_bb = L.append_block context "else" the_function in
280         add_terminal (stmt (L.builder_at_end context else_bb) else_stmt
                )
281           (L.build_br merge_bb);
282
283         ignore (L.build_cond_br bool_val then_bb else_bb builder);
284         L.builder_at_end context merge_bb
285
286       | S.SWhile (predicate, body) ->
287           let pred_bb = L.append_block context "while" the_function in
288           ignore (L.build_br pred_bb builder);
289
290           let body_bb = L.append_block context "while_body" the_function
                in
291           add_terminal (stmt (L.builder_at_end context body_bb) body)
292             (L.build_br pred_bb);
293
294           let pred_builder = L.builder_at_end context pred_bb in
295           let bool_val = expr pred_builder predicate in
296
297           let merge_bb = L.append_block context "merge" the_function in
298           ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder
                );
299           L.builder_at_end context merge_bb
300
301       | S.SFor (e1, e2, e3, body) -> stmt builder
302             ( S.SBlock [S.SExpr e1 ; S.SWhile (e2, S.SBlock [body ; S.
```

```
                SExpr e3]) ] )
303     in
304
305     (* Build the code for each statement in the function *)
306     let builder = stmt builder (S.SBlock fdecl.S.sbody) in
307
308     (* Add a return if the last block falls off the end *)
309     add_terminal builder (match fdecl.S.styp with
310         A.Void -> L.build_ret_void
311       | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
312   in
313
314   List.iter build_function_body functions;
315   the_module
```

8.3 inputPic.c

```
1  /*Part of our C library
2  Created by Gabriel Kramer-Garcia
3  */
4  #include <stdio.h>
5  #define new_min(x,y) ((x) <= (y)) ? (x) : (y)
6
7  void inputPic(int effect)
8  {
9    int pix_x=300,pix_y=300;     // image dimns in pixels
10   static int image[300][300][4]; // first [] number here is total pixels
         of each color in
11                                  // my image, 3 is for //RGB values
12   FILE *streamIn;
13   // //opening 24bit image
14   streamIn = fopen("edwards.bmp", "r"); // a bigger star in black and a
         smaller
15   //                                          // star in blue (refer
         figure attached)
16
17   int byte;
18   int i,j;
19   for(i=0;i<57;i++) {
20     byte = fgetc(streamIn);  // strip out BMP header-> for //24bit bmp
           image
21   }
22
23   // // initiating with new "i" different from above
24   int k;
25   for(k=0;k<pix_y;k++) {
26     for(j=0;j<pix_x;j++) {
27       image[k][j][3] = fgetc(streamIn);
28       image[k][j][2] = fgetc(streamIn);  // use BMP 24bit with no alpha
             channel
29       image[k][j][1] = fgetc(streamIn);  // BMP uses BGR but we want RGB
             , grab //byte-by-byte
30       image[k][j][0] = fgetc(streamIn);
31     }
32   }
33   if(effect==1){
34   int temp;
35   for(k=0;k<pix_y;k++) {
36     for(j=0;j<pix_x;j++) {
37       temp = (image[k][j][2] + image[k][j][1] + image[k][j][0])/3;
38       image[k][j][2] = temp;
39       image[k][j][1] = temp;
40       image[k][j][0] = temp;
41     }
42   }
43   }
44   if(effect==2){
45   float tempR;
46   float tempG;
47   float tempB;
48   for(k=0;k<pix_y;k++) {
49     for(j=0;j<pix_x;j++) {
50       tempR = image[k][j][2]*.189 + image[k][j][1]*.769 + image[k][j
             ][0]*.393;
51       tempG = image[k][j][2]*.168 + image[k][j][1]*.686 + image[k][j
             ][0]*.349;
52       tempB = image[k][j][2]*.131 + image[k][j][1]*.534 + image[k][j
             ][0]*.272;
```

```
53          image[k][j][2] = new_min((int) tempB, 255);
54          image[k][j][1] = new_min((int) tempG, 255);
55          image[k][j][0] = new_min((int) tempR, 255);
56        }
57      }
58      }
59      FILE *fp = fopen("outimage.ppm", "wb"); /* b - binary mode */
60      (void) fprintf(fp, "P6\n%d %d\n255\n", pix_x, pix_y);
61      for(k=0;k<pix_y;k++) {
62        for(j=0;j<pix_x;j++) {
63              static unsigned char color[3];
64              color[0] = image[k][j][0]; /* red */
65              color[1] = image[k][j][1];  /* green */
66              color[2] = image[k][j][2];  /* blue */
67              (void) fwrite(color, 1, 3, fp);
68          }
69      }
70      (void) fclose(fp);
71  }
```

8.4 Makefile

```
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

all : clean pixelman.native printbig.o makePic.o inputPic.o

pixelman.native :
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
	     pixelman.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log *.diff pixelman scanner.ml parser.ml parser.mli
	rm -rf printbig makePic inputPic
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe *.err

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx pixelman.cmx

pixelman : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o pixelman

scanner.ml : scanner.mll
	ocamllex scanner.mll

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

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

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

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<

# Testing the "printbig" example

printbig : printbig.c
	cc -o printbig -DBUILD_TEST printbig.c

makePic : makePic.c
	cc -o makePic -DBUILD_TEST makePic.c

inputPic : inputPic.c
	cc -o inputPic -DBUILD_TEST inputPic.c

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
```

```
57 ast.cmx :
58 codegen.cmo : ast.cmo
59 codegen.cmx : ast.cmx
60 pixelman.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
61 pixelman.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
62 parser.cmo : ast.cmo parser.cmi
63 parser.cmx : ast.cmx parser.cmi
64 scanner.cmo : parser.cmi
65 scanner.cmx : parser.cmx
66 semant.cmo : ast.cmo
67 semant.cmx : ast.cmx
68 parser.cmi : ast.cmo
69
70 # Building the tarball
71
72 TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3
       \
73     func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3
          \
74     hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2
          \
75     while1 while2 printbig
76
77 FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2
       \
78     for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8
          \
79     func9 global1 global2 if1 if2 if3 nomain return1 return2 while1
          \
80     while2
81
82 TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
83             $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)
84
85 TARFILES = ast.ml codegen.ml Makefile pixelman.ml parser.mly README
       scanner.mll \
86         semant.ml testall.sh $(TESTFILES:%=tests/%) printbig.c makePic.c
               inputPic.c arcade-font.pbm \
87         font2c
88
89 pixelman.tar.gz : $(TARFILES)
90         cd .. && tar czf pixelman/pixelman.tar.gz \
91                 $(TARFILES:%=pixelman/%)
```

8.5 makePic.c

```
1  /*
2  Part of our C library
3  Created by Gabriel Kramer-Garcia
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  void makePic(int width, int height, int red, int green, int blue){
10     const int dimx = width, dimy = height;
11     int i, j;
12     FILE *fp = fopen("first.ppm", "wb"); /* b - binary mode */
13     (void) fprintf(fp, "P6\n%d %d\n255\n", dimx, dimy);
14     for (j = 0; j < dimy; ++j)
15     {
16         for (i = 0; i < dimx; ++i)
17         {
18             static unsigned char color[3];
19             color[0] = red;  /* red */
20             color[1] = green;  /* green */
21             color[2] = blue;  /* blue */
22             (void) fwrite(color, 1, 3, fp);
23         }
24     }
25     (void) fclose(fp);
26 }
```

8.6 parser.mly

```
1  /* Ocamlyacc parser for Pixelman
2  *  Teresa Choe
3  *  Anthony Chan
4  *  Brian Tsau
5  *  Gabriel Kramer-Garcia
6  */
7
8  %{
9  open Ast
10 %}
11
12 %token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA COLON
       DOT
13 %token LMATBRACK RMATBRACK
14 %token PLUS MINUS TIMES DIVIDE ASSIGN NOT INTCAST FLOATCAST
15 %token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
16 %token RETURN IF ELSE FOR WHILE INT FLOAT BOOL VOID DEF STRING CHAR
       SIZEOF /*IMAGE*/
17
18 %token NOVECLBRACKET
19 %token BREAK CONTINUE
20 %token LSHIFT RSHIFT BITAND BITXOR BITOR MOD
21 %token <int> INT_LITERAL
22 %token <string> ID
23 %token <char> CHAR_LITERAL
24 %token <float> FLOAT_LITERAL
25 %token <string> STRING_LITERAL
26 %token EOF
27
28 %nonassoc NOELSE
29 %nonassoc ELSE
30 %nonassoc NOVECLBRACKET
31 %nonassoc LMATBRACK
32 %nonassoc LBRACKET
33 %nonassoc DOT
34 %right ASSIGN
35 %left OR
36 %left AND
37 %left BITOR
38 %left BITXOR
39 %left BITAND
40 %left EQ NEQ
41 %left LT GT LEQ GEQ
42 %left LSHIFT RSHIFT
43 %right INTCAST FLOATCAST
44 %left PLUS MINUS
45 %left TIMES DIVIDE MOD
46 %right NOT NEG
47
48 %start program
49 %type <Ast.program> program
50
51 %%
52
53 program:
54   decls EOF { $1 }
55
56 decls:
57    /* nothing */ { [], [] }
58  | decls vdecl { ($2 :: fst $1), snd $1 }
59  | decls fdecl { fst $1, ($2 :: snd $1) }
```

```
60
61 fdecl:
62    DEF typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
          RBRACE
63      { { typ = $2;
64          fname = $3;
65          formals = $5;
66          locals = List.rev $8;
67          body = List.rev $9 } }
68
69 formals_opt:
70     /* nothing */ { [] }
71   | formal_list   { List.rev $1 }
72
73 formal_list:
74     typ ID                   { [($1,$2)] }
75   | formal_list COMMA typ ID { ($3,$4) :: $1 }
76
77 typ:
78     INT { Int }
79   | BOOL { Bool }
80   | FLOAT { Float }
81   | CHAR { Char }
82   | STRING { String }
83   | VOID { Void }
84   /*| im_t { $1 }*/
85   | vec_t { $1 }
86   | mat_t { $1 }
87
88 vdecl_list:
89     /* nothing */    { [] }
90   | vdecl_list vdecl { $2 :: $1 }
91
92 vdecl:
93    typ ID SEMI { ($1, $2) }
94
95 vec_t:
96    typ LBRACKET expr RBRACKET %prec NOVECLBRACKET { Vector($1, $3) }
97
98 mat_t:
99    typ LBRACKET expr RBRACKET LBRACKET expr RBRACKET { Matrix($1, $3, $6
          ) }
100
101 /*im_t:
102    IMAGE LBRACKET expr COMMA expr RBRACKET %prec NOVECLBRACKET { Image(
          $3, $5) }*/
103
104
105 stmt_list:
106     /* nothing */  { [] }
107   | stmt_list stmt { $2 :: $1 }
108
109 stmt:
110     expr SEMI { Expr $1 }
111   | RETURN SEMI { Return Noexpr }
112   | RETURN expr SEMI { Return $2 }
113   | LBRACE stmt_list RBRACE { Block(List.rev $2) }
114   | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
115   | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
116   | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
117      { For($3, $5, $7, $9) }
118   | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
119   /* | BREAK SEMI { Break }
```

```
120   | CONTINUE SEMI { Continue } */
121
122 expr_opt:
123     /* nothing */ { Noexpr }
124   | expr          { $1 }
125
126 expr:
127   literals { $1 }
128   | ID               { Id($1) }
129   | expr PLUS   expr { Binop($1, Add,   $3) }
130   | expr MINUS  expr { Binop($1, Sub,   $3) }
131   | expr TIMES  expr { Binop($1, Mult,  $3) }
132   | expr DIVIDE expr { Binop($1, Div,   $3) }
133   | expr MOD    expr { Binop($1, Mod,   $3) }
134   | expr EQ     expr { Binop($1, Equal, $3) }
135   | expr NEQ    expr { Binop($1, Neq,   $3) }
136   | expr LT     expr { Binop($1, Less,  $3) }
137   | expr LEQ    expr { Binop($1, Leq,   $3) }
138   | expr GT     expr { Binop($1, Greater, $3) }
139   | expr GEQ    expr { Binop($1, Geq,   $3) }
140   | expr AND    expr { Binop($1, And,   $3) }
141   | expr OR     expr { Binop($1, Or,    $3) }
142   | expr LSHIFT expr { Binop($1, Shiftleft, $3) }
143   | expr RSHIFT     expr { Binop($1, Shiftright, $3) }
144   | expr BITAND     expr { Binop($1, Bitand, $3) }
145   | expr BITOR      expr { Binop($1, Bitor, $3) }
146   | expr BITXOR     expr { Binop($1, Bitxor, $3) }
147   | MINUS expr %prec NEG { Unop(Neg, $2) }
148   | NOT expr         { Unop(Not, $2) }
149   | INTCAST expr    { Unop(IntCast, $2) }
150   | FLOATCAST expr  { Unop(FloatCast, $2) }
151   | expr ASSIGN expr   { Assign($1, $3) }
152   | SIZEOF LPAREN ID RPAREN  { SizeOf($3) }
153   | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
154   | LPAREN expr RPAREN { $2 }
155   | ID LBRACKET expr RBRACKET { VecAccess($1, $3) }
156   | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET { MatAccess($1, $3,
        $6) }
157   | ID LBRACKET expr RBRACKET LBRACKET RBRACKET { MatRow($1, $3) }
158   | ID LBRACKET RBRACKET LBRACKET expr RBRACKET { MatCol($1, $5) }
159   /*| ID DOT LBRACKET INT_LITERAL RBRACKET { ImAccess($1, $4) }*/
160
161
162 primitive_literals:
163     INT_LITERAL      { Int_Literal($1) }
164   | STRING_LITERAL   { String_Literal($1) }
165   | FLOAT_LITERAL    { Float_Literal($1) }
166   | CHAR_LITERAL     { Char_Literal($1) }
167   | TRUE             { Bool_Literal(true) }
168   | FALSE            { Bool_Literal(false) }
169
170 literals:
171   primitive_literals { $1 }
172   | LBRACKET array_literal RBRACKET { Vector_Literal(List.rev $2) }
173   | LMATBRACK matrix_literal RMATBRACK { Matrix_Literal(List.rev $2) }
174
175 matrix_literal:
176   LBRACKET array_literal RBRACKET { [$2] }
177   | matrix_literal BITAND LBRACKET array_literal RBRACKET { $4 :: $1 }
178
179 array_literal:
180   primitive_literals { [$1] }
181   | array_literal COMMA primitive_literals { $3 :: $1 }
```

```
182
183 actuals_opt:
184     /* nothing */ { [] }
185   | actuals_list  { List.rev $1 }
186
187 actuals_list:
188     expr                      { [$1] }
189   | actuals_list COMMA expr { $3 :: $1 }
```

8.7 pixelman.ml

```
1    (*check the resulting AST, generate LLVM IR, and dump the module *)
2  (*Linking to our standard library
3  Gabriel Kramer-Garcia
4  *)
5  type action = Ast | LLVM_IR | Compile
6
7  let _ =
8    let action = if Array.length Sys.argv > 1 then
9      List.assoc Sys.argv.(1) [ ("-a", Ast);      (* Print the AST only *)
10                               ("-l", LLVM_IR);  (* Generate LLVM, don't
                                     check *)
11                               ("-c", Compile) ] (* Generate, check LLVM
                                     IR *)
12   else Compile in
13
14 let file_to_string file =
15     let array_string = ref [] in
16       let ic = file in
17           try
18              while true do
19                  array_string := List.append !array_string [input_line
                        ic]
20               done;
21               String.concat "\n" !array_string
22           with End_of_file -> close_in ic; String.concat "\n" !
                array_string
23
24   in
25   let in_file = open_in "stdlib.px" in
26   let string_in = file_to_string in_file in
27   let other_file = file_to_string stdin in
28   let str = String.concat "\n" [other_file; string_in] in
29
30
31   let lexbuf = Lexing.from_string str in
32   let ast = Parser.program Scanner.token lexbuf in
33   let sast = Semant.check ast in
34   match action with
35     Ast -> print_string (Ast.string_of_program ast)
36   | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate
       sast))
37   | Compile -> let m = Codegen.translate sast in
38     Llvm_analysis.assert_valid_module m;
39     print_string (Llvm.string_of_llmodule m)
```

8.8 printbig.c

```
1     /*
2   * A function illustrating how to link C code to code generated from
        LLVM
3   */
4
5  #include <stdio.h>
6
7  /*
8   * Font information: one byte per row, 8 rows per character
9   * In order, space, 0-9, A-Z
10  */
11 static const char font[] = {
12   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
13   0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
14   0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
15   0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
16   0x20, 0x61, 0x49, 0x4d, 0x4f, 0x7b, 0x31, 0x00,
17   0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
18   0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
19   0x3c, 0x7e, 0x4b, 0x49, 0x49, 0x79, 0x30, 0x00,
20   0x03, 0x03, 0x71, 0x79, 0x0d, 0x07, 0x03, 0x00,
21   0x36, 0x4f, 0x4d, 0x59, 0x59, 0x76, 0x30, 0x00,
22   0x06, 0x4f, 0x49, 0x49, 0x69, 0x3f, 0x1e, 0x00,
23   0x7c, 0x7e, 0x13, 0x11, 0x13, 0x7e, 0x7c, 0x00,
24   0x7f, 0x7f, 0x49, 0x49, 0x49, 0x7f, 0x36, 0x00,
25   0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
26   0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
27   0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
28   0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
29   0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
30   0x7f, 0x7f, 0x08, 0x08, 0x08, 0x7f, 0x7f, 0x00,
31   0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
32   0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
33   0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
34   0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x40, 0x00,
35   0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
36   0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
37   0x3e, 0x7f, 0x41, 0x41, 0x41, 0x7f, 0x3e, 0x00,
38   0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
39   0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
40   0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
41   0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
42   0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
43   0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
44   0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
45   0x1f, 0x7f, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
46   0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
47   0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
48   0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
49 };
50
51 void printbig(int c)
52 {
53   int index = 0;
54   int col, data;
55   if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;
56   else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;
57   do {
58     data = font[index++];
59     for (col = 0 ; col < 8 ; data <<= 1, col++) {
60       char d = data & 0x80 ? 'X' : ' ';
```

```
61          putchar(d); putchar(d);
62        }
63        putchar('\n');
64      } while (index & 0x7);
65  }
66
67  #ifdef BUILD_TEST
68  int main()
69  {
70    char s[] = "HELLO WORLD09AZ";
71    char *c;
72    for ( c = s ; *c ; c++) printbig(*c);
73  }
74  #endif
```

8.9 sast.ml

```
1    (* pixelman's Semantically Checked Abstract Syntax Tree and functions
         for printing it
2  * Gabriel Kramer-Garcia
3  * Anthony Chan
4  * Teresa Choe
5  * Brian Tsau
6  *)
7
8  open Ast
9
10 (* sexpressions, ssome swith sdatatype sas sadditional sinformation *)
11 type sexpr =
12     SInt_Literal of int
13   | SFloat_Literal of float
14   | SChar_Literal of char
15   | SString_Literal of string
16   | SBool_Literal of bool
17   | SVector_Literal of sexpr list * typ
18   | SMatrix_Literal of sexpr list list * typ
19   | SId of string * typ
20   | SBinop of sexpr * op * sexpr * typ
21   | SUnop of uop * sexpr * typ
22   | SAssign of sexpr * sexpr * typ
23   | SVecAccess of string * sexpr * typ
24   | SMatAccess of string * sexpr * sexpr * typ
25   (*| SImAccess of string * int * typ*)
26   | SCall of string * sexpr list * typ
27   | SSizeOf of string * typ
28   | SNoexpr
29
30 (* sstatements *)
31 type sstmt =
32     SBlock of sstmt list
33   | SExpr of sexpr
34   | SReturn of sexpr
35   | SIf of sexpr * sstmt * sstmt
36   | SFor of sexpr * sexpr * sexpr * sstmt
37   | SWhile of sexpr * sstmt
38   (* | SBreak
39   | SContinue  *)
40
41 (* sfunction sdeclarations *)
42 type sfunc_decl = {
43   styp : typ;
44   sfname : string;
45   sformals : bind list;
46   slocals : bind list;
47   sbody : sstmt list;
48 }
49
50 (* sprogram *)
51 type sprogram = bind list * func_decl list
```

8.10 scanner.mll

```
1    (* Ocamllex scanner for pixelman
2  * Brian Tsau
3  * Teresa Choe
4  * Gabriel Kramer-Garcia
5  * Anthony Chan
6  * *)
7
8  { open Parser }
9
10 let character = [' '-'!' '#'-'[' ']'-'~'] | ('\\' ['\\' ''' '"' 'n' 'r'
       't'])
11 let digit = ['0'-'9']
12 let whitespace = [' ' '\t' '\n' '\r']
13 let float_lit = (digit*) ['.'] digit+
14 let int_lit = digit+
15
16 rule token = parse
17   whitespace { token lexbuf }              (* Whitespace *)
18   | ":)"     { sl_comment lexbuf }
19   | "(:"     { comment lexbuf }
20
21   (* Separators *)
22 | '('       { LPAREN }
23 | ')'       { RPAREN }
24 | '{'       { LBRACE }
25 | '}'       { RBRACE }
26 | ';'       { SEMI }
27 | ','       { COMMA }
28 | "["       { LBRACKET }
29 | "]"       { RBRACKET }
30 | "[|"      { LMATBRACK }
31 | "|]"      { RMATBRACK }
32 (*| ':'       { COLON } *)
33 | "."       { DOT }
34
35         (* Assignment Operators *)
36 | '='       { ASSIGN }
37
38         (* Casting Operators *)
39 | "$int"     { INTCAST }
40 | "$float"   { FLOATCAST }
41
42     (* Binary Arithmetic Operators *)
43 | '+'       { PLUS }
44 | '-'       { MINUS }
45 | '*'       { TIMES }
46 | '/'       { DIVIDE }
47 | "%"       { MOD }
48
49         (* Binary Comparison Operators *)
50 | "=="      { EQ }
51 | "!="      { NEQ }
52 | '<'       { LT }
53 | "<="      { LEQ }
54 | ">"       { GT }
55 | ">="      { GEQ }
56
57         (* Binary Boolean Operators *)
58 | "&&"      { AND }
59 | "||"      { OR }
60
```

```
61          (* Unary Boolean Operators *)
62  | "!"       { NOT }
63
64          (* Binary Bitwise Operators *)
65  | "|"       { BITOR }
66  | "<<"      { LSHIFT }
67  | ">>"      { RSHIFT }
68  | "&"       { BITAND }
69  | "^"       { BITXOR }
70
71          (* Branching Control *)
72  | "if"        { IF }
73  | "else"      { ELSE }
74  | "for"       { FOR }
75  | "while"     { WHILE }
76  (* | "continue" { CONTINUE }
77  | "break"     { BREAK } *)
78  | "return"    { RETURN }
79
80          (* function definition *)
81  | "def"       { DEF }
82
83          (* Data and Return Types *)
84  | "char"    { CHAR }
85  | "int"     { INT }
86  | "float"   { FLOAT }
87  | "bool"    { BOOL }
88  | "string"  { STRING }
89  | "void"    { VOID }
90  | "true"    { TRUE }
91  | "false"   { FALSE }
92  (*| "Image"   { IMAGE }*)
93  | "sizeof"  { SIZEOF }
94
95  (* Literals *)
96  | int_lit    as lxm     { INT_LITERAL(int_of_string lxm) }
97  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
98  | ''' character '''     as lxm     { CHAR_LITERAL(lxm.[1]) }
99  | '"' ((character*) as lxm) '"' { STRING_LITERAL(lxm) }
100 | float_lit as lxm    { FLOAT_LITERAL(float_of_string lxm) }
101
102 (* Comment *)
103 | eof { EOF }
104 | _ as char { raise (Failure("illegal character " ^ Char.escaped char))
      }
105
106 and comment = parse
107   ":)" { token lexbuf }
108   | _  { comment lexbuf }
109
110 and sl_comment = parse
111   | '\n' { token lexbuf }
112   | _    { sl_comment lexbuf }
```

8.11 semant.ml

```
1    (* Semantic checking for the pixelman compiler
2  * Teresa Choe
3  * Brian Tsau
4  * Anthony Chan
5  * Gabriel Kramer-Garcia
6  *)
7
8  open Ast
9  open Sast
10
11 module StringMap = Map.Make(String)
12
13 (* Semantic checking of a program. Returns void if successful,
14    throws an exception if something is wrong.
15
16    Check each global variable, then check each function *)
17
18 let check (globals, functions) =
19
20   (* Raise an exception if the given list has a duplicate *)
21   let report_duplicate exceptf list =
22     let rec helper = function
23         n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
24       | _ :: t -> helper t
25       | [] -> ()
26     in helper (List.sort compare list)
27   in
28
29   (* Raise an exception if a given binding is to a void type *)
30   let check_not_void exceptf = function
31       (Void, n) -> raise (Failure (exceptf n))
32     | _ -> ()
33   in
34
35   (**** Checking Global Variables ****)
36
37   List.iter (check_not_void (fun n -> "illegal void global " ^ n))
       globals;
38
39   report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd
       globals);
40
41   (**** Checking Functions ****)
42
43   let protected_functions = ["print_int"; "perror"; "scan"; "size"; "
       load"; "write";
44                              "display"; "resize"; "transform"; "
                                   print_float"; "print_string"] in
45   let rec check_protected = function
46     [] -> ()
47     | h :: t -> if List.mem h (List.map (fun fd -> fd.fname) functions)
48         then raise (Failure ("function" ^ h ^ "may not be defined"))
49         else ignore (check_protected t)
50   in check_protected protected_functions;
51
52   report_duplicate (fun n -> "duplicate function " ^ n)
53     (List.map (fun fd -> fd.fname) functions);
54
55   (* Function declaration for a named function *)
56   let built_in_decls =  StringMap.add "print_int"
57      { typ = Void; fname = "print_int"; formals = [(Int, "x")];
```

```
58        locals = []; body = [] } (StringMap.add "printb"
59      { typ = Void; fname = "printb"; formals = [(Bool, "x")];
60        locals = []; body = [] } (StringMap.add "print_newline"
61      { typ = Void; fname = "print_newline"; formals = [];
62        locals = []; body = [] } (StringMap.add "printbig"
63      { typ = Void; fname = "printbig"; formals = [(Int, "x")];
64        locals = []; body = [] } (StringMap.add "inputPic"
65      { typ = Void; fname = "inputPic"; formals = [(Int, "x")];
66        locals = []; body = [] } (StringMap.add "makePic"
67      { typ = Void; fname = "makePic"; formals = [(Int, "x");(Int, "x");(
          Int, "x");(Int, "x");(Int, "x")];
68        locals = []; body = [] } (StringMap.add "print_string"
69      { typ = Void; fname = "print_string"; formals = [(String, "x")];
70        locals = []; body = [] } (StringMap.singleton "print_float"
71      { typ = Void; fname = "print_float"; formals = [(Float, "x")];
72        locals = []; body = [] } )))))))
73    in
74
75    let function_decls = List.fold_left (fun m fd -> StringMap.add fd.
        fname fd m)
76                         built_in_decls functions
77    in
78
79    let function_decl s = try StringMap.find s function_decls
80         with Not_found -> raise (Failure ("unrecognized function " ^ s))
81    in
82
83    let _ = function_decl "main" in (* Ensure "main" is defined *)
84
85    let fdecl_to_sfdecl func =
86
87      List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
88        " in " ^ func.fname)) func.formals;
89
90      report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.
          fname)
91        (List.map snd func.formals);
92
93      report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.
          fname)
94        (List.map snd func.locals);
95
96      (* Type of each variable (global, formal, or local) *)
97      let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
98          StringMap.empty (globals @ func.formals @ func.locals )
99      in
100
101     let type_of_identifier s =
102       try StringMap.find s symbols
103       with Not_found -> raise (Failure ("undeclared identifier " ^ s))
104     in
105
106     let access_type = function
107       Vector(t, _) -> t
108       | Matrix(t, _, _) -> t
109       | _ -> raise (Failure ("illegal matrix/vector access"))
110     in
111
112     let is_vec_matrix t = match t with
113         Vector(_,_) -> ()
114       | Matrix(_,_,_) -> ()
115       | _ -> raise (Failure ("cannot get size of non-vector/non-matrix
            type"))
```

```
116     in
117
118     let get_sexpr_type se = match se with
119         SInt_Literal(_) -> Int
120         | SFloat_Literal(_) -> Float
121         | SChar_Literal(_) -> Char
122         | SString_Literal(_) -> String
123         | SBool_Literal(_) -> Bool
124         | SVector_Literal(_, t) -> t
125         | SMatrix_Literal(_, t) -> t
126         | SId(_,t) -> t
127         | SBinop(_,_,_,t) -> t
128         | SUnop(_,_,t) -> t
129         | SAssign(_,_,t) -> t
130         | SVecAccess(_,_,t) -> t
131         | SMatAccess(_,_,_,t) -> t
132         (*| SImAccess(_,_,t) -> t*)
133         | SCall(_,_,t) -> t
134         | SSizeOf(_,t) -> t
135         | SNoexpr -> Void
136     in
137
138     let get_binop_boolean_sexpr se1 se2 op =
139       let t1 = get_sexpr_type se1 in
140       let t2 = get_sexpr_type se2 in
141       match (t1, t2) with
142         (Bool, Bool) -> SBinop(se1, op, se2, Bool)
143         | _ -> raise (Failure ("can only perform boolean operators with
              Int/Bool types"))
144
145     and get_unop_boolean_sexpr se op =
146       let t = get_sexpr_type se in
147       match t with
148         Bool -> SUnop(op, se, Bool)
149         | _ -> raise (Failure ("can only perform boolean operators with
              Int/Bool types"))
150
151     and get_binop_arithmetic_sexpr se1 se2 op =
152       let t1 = get_sexpr_type se1 in
153       let t2 = get_sexpr_type se2 in
154       match (t1, t2) with
155         (Int, Int) -> SBinop(se1, op, se2, Int)
156         | (Int, Float) -> SBinop(SUnop(FloatCast, se1, Float), op, se2,
              Float)
157         | (Float, Int) -> SBinop(se1, op, SUnop(FloatCast, se2, Float),
              Float)
158         | (Float, Float) -> SBinop(se1, op, se2, Float)
159         | (Vector(tm1, Int_Literal(i)), ta2) -> (match op with
160               Mult -> (match ta2 with
161                   Float -> SCall("scalar_mult_vecf", [se2; se1], Vector(
                        Float, Int_Literal(i)))
162                   | Int -> if tm1 == Float
163                     then SCall("scalar_mult_vecf", [se2; se1], Vector(
                          Float, Int_Literal(i)))
164                     else SCall("scalar_mult_veci", [se2; se1], Vector(
                          Int, Int_Literal(i)))
165                   | Vector(tm2, _) -> if tm2 == Float || tm1 == Float
166                       then SCall("vec_dot_productf", [se2; se1], Float)
167                       else SCall("vec_dot_producti", [se2; se1], Int)
168                   | Matrix(tm2, _, Int_Literal(j2)) -> if tm2 == Float
                        || tm1 == Float
169                           then SCall("vec_mat_multf", [se1; se2], Vector
                              (Float, Int_Literal(j2)))
```

```
170                         else SCall("vec_mat_multi", [se1; se2], Vector
                                (Int, Int_Literal(j2)))
171                 | _ -> raise (Failure ("can only perform binary
                        arithmetic operators with Int/Float variables or
                        matrices"))
172             )
173             | Sub -> (match ta2 with
174                     Vector(tm2, Int_Literal(i)) -> if tm2 == Float ||
                           tm1 == Float
175                       then SCall("vec_subf", [se1; se2], Vector(Float,
                           Int_Literal(i)))
176                       else SCall("vec_subi", [se1; se2], Vector(Int,
                           Int_Literal(i)))
177                       | _ -> raise (Failure ("oh no! can only perform
                           this operation on vector of same length.")))
178             | Add -> (match ta2 with
179                     Vector(tm2, Int_Literal(i)) -> if tm2 == Float ||
                           tm1 == Float
180                       then SCall("vec_vec_addf", [se1; se2], Vector(
                           Float, Int_Literal(i)))
181                       else SCall("vec_vec_addi", [se1; se2], Vector(Int,
                            Int_Literal(i)))
182                   | _ -> raise (Failure ("oh no! can only perform this
                            operation on vector of same length.")))
183             | _ -> raise (Failure ("oh no! cannot perform this
                    operation on vector.")))
184         | (Matrix(tm1, Int_Literal(i), Int_Literal(j)), ta2) -> (match
                op with
185             Mult -> (match ta2 with (* matrix x ta2 check *)
186                     Float -> if i == 3 then
187                       SCall("scalar_mult_mat3f", [se2; se1], Matrix(
                           Float, Int_Literal(i), Int_Literal(j)))
188                       else SCall("scalar_mult_mat2f", [se2; se1],
                           Matrix(Float, Int_Literal(i), Int_Literal(j)
                           ))
189                       | Int ->
190                       let sc_mat_f_fname = if i == 2 then "
                           scalar_mult_mat2f" else "scalar_mult_mat3f"
                           in
191                       let sc_mat_i_fname = if i == 2 then "
                           scalar_mult_mat2i" else "scalar_mult_mat3i"
                           in
192                       if tm1 == Int
193                         then SCall(sc_mat_i_fname, [se2; se1], Matrix(
                             Int, Int_Literal(i), Int_Literal(j)))
194                         else SCall(sc_mat_f_fname, [se2; se1], Matrix(
                             Float, Int_Literal(i), Int_Literal(j)))
195                       | Vector(tm2, _) -> if tm2 == Float || tm1 ==
                           Float
196                         then SCall("mat_vec_multf", [se1; se2], Vector
                             (Float, Int_Literal(i)))
197                         else SCall("mat_vec_multi", [se1; se2], Vector
                             (Int, Int_Literal(i)))
198                       | Matrix(tm2, _, Int_Literal(j2)) -> if tm2 ==
                           Float || tm1 == Float
199                         then SCall("mat_mat_multf", [se1; se2], Matrix
                             (Float, Int_Literal(i), Int_Literal(j2)))
200                         else SCall("mat_mat_multi", [se1; se2], Matrix
                             (Int, Int_Literal(i), Int_Literal(j2)))
201                       | _ -> raise (Failure ("can only perform binary
                           arithmetic operators with Int/Float variables
                           or matrices")))
202             | Sub -> (match ta2 with
```

```
203                     Matrix(tm2, Int_Literal(i), Int_Literal(j)) -> if tm2
                            == Float || tm1 == Float
204                       then SCall("mat_subf", [se1; se2], Matrix(Float,
                              Int_Literal(i), Int_Literal(j)))
205                       else SCall("mat_subi", [se1; se2], Matrix(Int,
                              Int_Literal(i), Int_Literal(j)))
206                     | _ -> raise (Failure ("oh no! can only perform this
                             operation on matrix of same size.")))
207                 | Add -> (match ta2 with
208                     Matrix(tm2, Int_Literal(i), Int_Literal(j)) -> if tm2
                            == Float || tm1 == Float
209                       then SCall("mat_addf", [se1; se2], Matrix(Float,
                              Int_Literal(i), Int_Literal(j)))
210                       else SCall("mat_addi", [se1; se2], Matrix(Int,
                              Int_Literal(i), Int_Literal(j)))
211                       | _ -> raise (Failure ("oh no! can only perform this
                              operation on matrix of same size.")))
212                 | _ -> raise (Failure ("oh no! cannot perform this
                        operation on matrix.") ))
213           | (Int, Vector(Int,Int_Literal(i))) -> SCall("scalar_mult_veci",
                [se1; se2], Vector(Int, Int_Literal(i)))
214           | (Float, Vector(Int,Int_Literal(i))) | (Int, Vector(Float,
                Int_Literal(i))) | (Float, Vector(Float,Int_Literal(i))) ->
                SCall("scalar_mult_vecf", [se1; se2], Vector(Float,
                Int_Literal(i)))
215           | (Int, Matrix(Int,Int_Literal(i),Int_Literal(j))) ->
216             if i == 2 then SCall("scalar_mult_mat2i", [se1; se2], Matrix
                    (Int, Int_Literal(i), Int_Literal(j)))
217             else SCall("scalar_mult_mat3i", [se1; se2], Matrix(Int,
                    Int_Literal(i), Int_Literal(j)))
218           | (Float, Matrix(Int,Int_Literal(i),Int_Literal(j))) -> if i ==
                2 then SCall("scalar_mult_mat2f", [se1; se2], Matrix(Float,
                Int_Literal(i), Int_Literal(j)))
219             else SCall("scalar_mult_mat3f", [se1; SCall("
                    mat_int_to_float", [se2], Matrix(Float,Int_Literal(i),
                    Int_Literal(j)))], Matrix(Float, Int_Literal(i),
                    Int_Literal(j)))
220           | (Int, Matrix(Float,Int_Literal(i),Int_Literal(j))) -> if i ==
                2 then SCall("scalar_mult_mat2f", [se1; se2], Matrix(Float,
                Int_Literal(i), Int_Literal(j)))
221             else SCall("scalar_mult_mat3f", [SUnop(FloatCast, se1, Float
                    ); se2], Matrix(Float, Int_Literal(i), Int_Literal(j)))
222           | (Float, Matrix(Float,Int_Literal(i),Int_Literal(j))) -> if i
                == 2 then SCall("scalar_mult_mat2f", [se1; se2], Matrix(
                Float, Int_Literal(i), Int_Literal(j)))
223             else SCall("scalar_mult_mat3f", [se1; se2], Matrix(Float,
                    Int_Literal(i), Int_Literal(j)))
224
225           | _ -> raise (Failure ("can only perform binary arithmetic
                operators with Int/Float variables or matrices"))
226
227       and get_unop_arithmetic_sexpr se op =
228         let t = get_sexpr_type se in
229         match t with
230           Int  -> SUnop(op, se, Int)
231           | Float -> SUnop(op, se, Float)
232           | Vector(Int,_) -> SCall("negVectori",[se],t)
233           | Vector(Float,_) -> SCall("negVectorf",[se],t)
234           | Matrix(Int,_,_) -> SCall("negMatrixi",[se],t)
235           | Matrix(Float,_,_) -> SCall("negMatrixf",[se],t)
236           | _ -> raise (Failure ("can only perform unary arithmetic
                operators with Int/Float variables or matrices"))
237
```

```
238     and get_binop_bitwise_sexpr se1 se2 op =
239       let t1 = get_sexpr_type se1 in
240       let t2 = get_sexpr_type se2 in
241       match (t1, t2) with
242         (Int, Int) -> SBinop(se1, op, se2, Int)
243         | _ -> raise (Failure ("can only perform bitwise operations on
              integer types"))
244
245     and get_binop_comparison_sexpr se1 se2 op =
246       let t1 = get_sexpr_type se1 in
247       let t2 = get_sexpr_type se2 in
248       match (t1, t2) with
249         (Int, Int) -> SBinop(se1, op, se2, Bool)
250         | (Float, Float) -> SBinop(se1, op, se2, Bool)
251         | _ -> raise (Failure ("can only compare ints/floats with
              themselves for inequalities"))
252
253     and get_unop_cast_sexpr se op =
254       let t1 = get_sexpr_type se in
255       let t = match t1 with
256         Int | Float -> t1
257         | Vector(t2,_) -> t2
258         | Matrix(t2,_,_) -> t2
259         | _ -> raise (Failure ("can only cast int/float expressions to
              int/float expressions"))
260       in
261       let op_t = match op with
262         IntCast -> Int
263         | FloatCast -> Float
264         | _ -> raise (Failure ("this is impossible to reach :~)"))
265       in
266       if t == op_t then se else
267         match t1 with
268           Int  -> SUnop(op, se, op_t)
269           | Float -> SUnop(op, se, op_t)
270           | Vector(Int,se1) -> SCall("vec_int_to_float", [se], Vector(
                op_t,se1))
271           | Vector(Float,se1) -> SCall("vec_float_to_int", [se], Vector(
                op_t,se1))
272           | Matrix(Int,se1,se2) -> SCall("mat_int_to_float", [se],
                Matrix(op_t,se1,se2))
273           | Matrix(Float,se1,se2) -> SCall("mat_float_to_int", [se],
                Matrix(op_t,se1,se2))
274           | _ -> raise (Failure ("this is impossible to reach :~)"))
275
276     and get_equality_type se1 se2 op =
277       let t1 = get_sexpr_type se1 in
278       let t2 = get_sexpr_type se2 in
279       match (t1, t2) with
280         (Int, Int) -> SBinop(se1, op, se2, Bool)
281         | (Float, Float) -> SBinop(se1, op, se2, Bool)
282         | (Char, Char) -> SBinop(se1, op, se2, Bool)
283         | _ -> raise (Failure ("can only compare ints/floats/chars with
              themselves for equality"))
284
285     (*and check_image_accessor ia =
286         match ia with
287         0 -> ()
288         | 1 -> ()
289         | 2 -> ()
290         | _ -> raise (Failure ("can only access 0, 1, or 2"))*)
291
292     (*and check_is_image id =
```

```
293         let idd = type_of_identifier id in
294         match idd with
295         Image(_,_) -> ()
296         | _ -> raise (Failure ("can only access images"))
297     *)
298
299     in
300
301     (* Return an sexpr given an expr *)
302     let rec expr_to_sexpr = function
303         Int_Literal(i) -> SInt_Literal(i)
304       | String_Literal(s) -> SString_Literal(s)
305       | Float_Literal(f) -> SFloat_Literal(f)
306       | Bool_Literal(b) -> SBool_Literal(b)
307       | Char_Literal(c) -> SChar_Literal(c)
308       | Vector_Literal(el) -> check_vector_types el
309       | Matrix_Literal(ell) -> check_matrix_types ell
310       | Id s -> SId(s, type_of_identifier s)
311       | SizeOf(s) -> is_vec_matrix (type_of_identifier s); SSizeOf(s,
            Int)
312       | VecAccess(v, e) -> check_int_expr e; check_vec_access_type v;
            SVecAccess(v, expr_to_sexpr e, access_type (type_of_identifier
             v))
313       | MatAccess(v, e1, e2) ->  check_int_expr e1; check_int_expr e2;
            check_mat_access_type v; SMatAccess(v, expr_to_sexpr e1,
            expr_to_sexpr e2, access_type (type_of_identifier v))
314       | MatRow(s,e) -> check_int_expr e; check_mat_access_type s;
            get_mat_row_sexpr s e
315       | MatCol(s,e) -> check_int_expr e; check_mat_access_type s;
            get_mat_col_sexpr s e
316       (*| ImAccess(idd, color) -> ignore(check_image_accessor color);
            ignore(check_is_image idd); SImAccess(idd,color,(
            type_of_identifier idd))*)
317       | Binop(e1, op, e2) (* as e *) -> get_binop_sexpr e1 e2 op
318       | Unop(op, e) (* as ex *) -> get_unop_sexpr op e
319       | Noexpr -> SNoexpr
320       | Assign(var, e) (* as ex *) -> get_assign_sexpr var e
321       (* | Call() *) (* let's put the std library functions in here *)
322       | Call(fname, actuals) as call -> let fd = function_decl fname in
323         if List.length actuals != List.length fd.formals then
324           raise (Failure ("expecting " ^ string_of_int
325             (List.length fd.formals) ^ " arguments in " ^
                 string_of_expr call))
326         else
327            SCall(fname,List.map2 (fun (ft, _) e -> let se =
328             let e' = expr_to_sexpr e (* some implicit int -> float
                    conversion done here *)
329             in let et2 = get_sexpr_type e'
330             in match ft with
331               Float -> if get_sexpr_type e' == Int then SUnop(
                    FloatCast, e', Float) else e'
332             | Vector(Float,_) -> (match et2 with
333                 Vector(Float,_) -> e'
334               | Vector(Int,Int_Literal(_))   -> (expr_to_sexpr (Unop
                    (FloatCast, e)))
335               | _                           -> raise (Failure("can
                    only have vector of int/float")))
336             | Matrix(Float,_,_) -> (match et2 with
337                 Matrix(Float,_,_) -> e'
338               | Matrix(Int,Int_Literal(_),Int_Literal(_)) -> (
                    expr_to_sexpr (Unop(FloatCast, e)))
339               | _                           -> raise (Failure("can
                    only have vector of int/float")))
```

```
340                 | _ -> e'
341               in
342                 let et = get_sexpr_type se in
343                   if check_formal_actual_call ft et then se else raise (
                          Failure ("illegal actual argument found " ^
344                       string_of_typ et ^ " expected " ^ string_of_typ ft ^ "
                              in " ^
345                       string_of_expr e))) fd.formals actuals
346               , fd.typ)
347
348       and check_formal_actual_call ft et = match ft with
349         Vector(t1,_) -> (match et with
350             Vector(t2,_) -> check_formal_actual_call t1 t2
351           | _                        -> false)
352         | Matrix(t1,_,_) -> (match et with
353             Matrix(t2,_,_) -> check_formal_actual_call t1 t2
354           | _              -> false)
355         | Float            -> et == Int || et == Float
356         | _                -> ft == et
357
358       and check_vec_access_type v =
359         let t = type_of_identifier v
360         in match t with
361           Vector(_,_) -> ()
362           | _ -> raise(Failure("cannot perform vector access on variable "
                    ^ v))
363
364       and check_mat_access_type v =
365         let t = type_of_identifier v
366         in match t with
367           Matrix(_,_,_) -> ()
368           | _ -> raise(Failure("cannot perform matrix access on variable "
                    ^ v))
369
370       and get_mat_row_sexpr v e =
371         let se = expr_to_sexpr e
372         in let t = type_of_identifier v
373         in match t with
374           Matrix(Int,_,Int_Literal(j)) -> SCall("get_mat_rowi", [SId(v,t);
                  se], Vector(Int,Int_Literal(j)))
375           | Matrix(Float,_,Int_Literal(j)) -> SCall("get_mat_rowf", [SId(v
                 ,t); se], Vector(Float,Int_Literal(j)))
376           | _ -> raise(Failure("cannot get row of non-matrix type"))
377
378
379       and get_mat_col_sexpr v e =
380         let se = expr_to_sexpr e
381         in let t = type_of_identifier v
382         in match t with
383           Matrix(Int,Int_Literal(i),_) -> SCall("get_mat_coli", [SId(v,t);
                  se], Vector(Int,Int_Literal(i)))
384           | Matrix(Float,Int_Literal(i),_) -> SCall("get_mat_colf", [SId(v
                 ,t); se], Vector(Float,Int_Literal(i)))
385           | _ -> raise(Failure("cannot get row of non-matrix type"))
386
387       (* only gets type of vector; does not go through whole vector all
             the time *)
388       and get_vec_type el = match el with
389           Int_Literal(_) :: ss -> get_vec_type ss
390           | Float_Literal(_) :: _ -> Float
391           | [] -> Int
392           | _ -> raise (Failure ("vector/matrix literals can only contain
                  float/int literals"))
```

```
393
394     and check_vector_types el =
395       let t = get_vec_type el in
396       let check_vec_el e = match e with (* this time check the whole
              vector and convert *)
397         Int_Literal(i) -> if t == Int then SInt_Literal(i) else
              SFloat_Literal(float i)
398       | Float_Literal(i) -> SFloat_Literal(i)
399       | _ -> raise (Failure ("vector/matrix literals can only contain
              float/int literals"))
400       in
401       SVector_Literal(List.map check_vec_el el, Vector(t, Int_Literal(
              List.length el)))
402
403     and check_matrix_types ell =
404       let check_row_lengths ell = (* check row lengths *)
405         let length_first_list = List.length (List.hd ell) in
406         List.iter (fun l -> if ((List.length l) != length_first_list)
              then
407                                 raise (Failure ("matrix row lengths must
                                    be equal")) else ()) ell
408       in
409       let get_mat_type ell =
410         let rec check_row_types el = match el with
411           Int_Literal(_) :: ss -> check_row_types ss
412         | Float_Literal(_) :: _ -> Float_Literal(0.0)
413         | [] -> Int_Literal(0)
414         | _ -> raise (Failure ("vector/matrix literals can only
                contain float/int literals"))
415         in
416         get_vec_type (List.map check_row_types ell)
417       in
418       let t = get_mat_type ell in
419       let mat_row_to_smat_row el =
420         let mat_el_to_smat_el e = match e with
421           Int_Literal(i) -> if t == Int then SInt_Literal(i) else
                SFloat_Literal(float i)
422         | Float_Literal(f) -> SFloat_Literal(f)
423         | _ -> raise (Failure ("vector/matrix literals can only
                contain float/int literals"))
424         in
425         List.map mat_el_to_smat_el el
426       in
427       check_row_lengths ell; SMatrix_Literal(List.map
              mat_row_to_smat_row ell, Matrix(t, Int_Literal(List.length ell
              ), Int_Literal(List.length (List.hd ell))))
428
429     and compare_vector_matrix_type v1 v2 =
430       match v1 with
431         | Vector(ty1, _) ->
432                 ( match v2 with
433                   Vector(ty2, _) -> ty1 == ty2 || ((ty1 == Float) && (
                          ty2 == Int))
434                   | _ -> raise (Failure ("cannot compare vectors and
                          matrices")) )
435         | Matrix(ty1, _, _) ->
436                 ( match v2 with
437                   Matrix(ty2, _, _) ->
438                           ty1 == ty2 || ((ty1 == Float) && (ty2 == Int))
439                   | _ -> raise (Failure ("cannot compare vectors and
                          matrices")) )
440         | _ -> raise (Failure ("matrix and vector dimensions must be int
                literals"))
```

```
441
442     and get_binop_sexpr e1 e2 op =
443       let se1 = expr_to_sexpr e1 in
444       let se2 = expr_to_sexpr e2 in
445       match op with
446         Equal | Neq -> get_equality_type se1 se2 op
447         | And | Or  -> get_binop_boolean_sexpr se1 se2 op
448         | Less | Leq | Greater | Geq -> get_binop_comparison_sexpr se1
              se2 op
449         | Shiftleft | Shiftright | Bitand | Bitor | Bitxor ->
              get_binop_bitwise_sexpr se1 se2 op
450         | Add | Sub | Mult | Div | Mod -> get_binop_arithmetic_sexpr se1
              se2 op
451
452     and get_unop_sexpr op e =
453       let se = expr_to_sexpr e in
454       match op with
455         Neg -> get_unop_arithmetic_sexpr se op
456         | Not -> get_unop_boolean_sexpr se op
457         | IntCast | FloatCast -> get_unop_cast_sexpr se op
458
459     and get_assign_sexpr e1 e2 =
460       let se1 = match e1 with
461         Id(_) | VecAccess(_,_) | MatAccess(_,_,_) -> expr_to_sexpr e1
462         | _ -> raise (Failure ("can only assign to variable or vector/
              matrix element"))
463       in
464       let se2 = expr_to_sexpr e2 in
465       let lt = get_sexpr_type se1 in
466       let rt = get_sexpr_type se2 in
467       match lt with
468       Vector(_,_) -> if compare_vector_matrix_type lt rt then SAssign(
            se1,se2,lt) else raise (Failure ("illegal assignment "))
469         | Matrix(Int,_,_) -> if compare_vector_matrix_type lt rt then
              SAssign(se1,se2,lt) else raise (Failure ("illegal assignment
              "))
470         | Matrix(Float,_,_) -> if compare_vector_matrix_type lt rt then
              (match rt with
471              Matrix(Int,_,_) -> SAssign(se1,expr_to_sexpr (Unop(FloatCast
                   , e2)),lt)
472              | _ -> (SAssign(se1,se2,lt)))
473           else raise (Failure ("illegal assignment "))
474         | _ -> if lt == Float && rt == Int
475                then SAssign(se1, (expr_to_sexpr (Unop(FloatCast, e2))),
                       Float)
476                else (if lt == rt then SAssign(se1,se2,lt)
477                      else raise (Failure ("illegal assignment " ^
478                      string_of_typ lt ^ " = " ^ string_of_typ rt ^
479                      " in: " ^ string_of_expr e1 ^ " = " ^
                         string_of_expr e2)))
480
481     and check_bool_expr e = if get_sexpr_type (expr_to_sexpr e) != Bool
482       then raise (Failure ("expected boolean expression in " ^
            string_of_expr e))
483       else ()
484
485     and check_int_expr e = if get_sexpr_type (expr_to_sexpr e) != Int
486       then raise (Failure ("expected integer expression in " ^
            string_of_expr e))
487       else ()
488
489     and check_int_literal_expr e = match e with
490       Int_Literal(_) -> ()
```

```
491       | _ -> raise(Failure("can only declare vectors/matrices/images
              with int literals"))
492     in
493
494     (* Return a sstmt given a stmt *)
495     let rec stmt_to_sstmt = function
496             Block(sl) -> let rec check_block = function (* just check if
                    return statement is end of block *)
497           [Return _] -> ()
498         | Return _ :: _ -> raise (Failure "nothing may follow a return
                ")
499         | Block sl :: ss -> check_block (sl @ ss)
500         | _ :: ss -> (* stmt_to_sstmt s; *) check_block ss
501         | [] -> ()
502       in
503       check_block sl; SBlock(List.map stmt_to_sstmt sl)
504       | Expr(e) -> SExpr(expr_to_sexpr e)
505       | Return(e) -> let se = expr_to_sexpr e in
506       let t = get_sexpr_type se in
507       if check_formal_actual_call t func.typ then SReturn(se) else
508        raise (Failure ("return gives " ^ string_of_typ t ^ " expected
              " ^
509                        string_of_typ func.typ ^ " in " ^
                           string_of_expr e))
510       | If(p, b1, b2) -> check_bool_expr p; SIf(expr_to_sexpr p,
            stmt_to_sstmt b1, stmt_to_sstmt b2)
511       | For(e1, e2, e3, st) -> SFor(expr_to_sexpr e1, expr_to_sexpr e2,
            expr_to_sexpr e3, stmt_to_sstmt st)
512       | While(p, s) -> check_bool_expr p; SWhile(expr_to_sexpr p,
            stmt_to_sstmt s)
513       (* | Break -> SBreak
514       | Continue -> SContinue *)
515     in
516
517     (* check variable declaration type *)
518     let check_var_decl (t, id) = match t with
519       Int | Bool | Float | Char | String -> (t, id)
520       | Void -> raise (Failure ("cannot declare a void type variable"))
521       | Vector(t1, e) -> check_int_literal_expr e;
522       if (t1 != Float) && (t1 != Int)
523       then raise(Failure("can only have vectors/matrices of ints/
              floats"))
524       else (); (t, id)
525       | Matrix(t1, e1, e2) -> check_int_literal_expr e1;
            check_int_literal_expr e2;
526       if (t1 != Float) && (t1 != Int)
527       then raise(Failure("can only have vectors/matrices of ints/
              floats"))
528       else (); (t, id)
529       (*| Image(h,w) -> check_int_literal_expr h; check_int_literal_expr
            w; (t, id)*)
530     in
531
532     let check_formal_bind (t, id) = match t with
533       Vector(_,e) -> if e != Noexpr then check_int_literal_expr e else
              (); (t, id)
534       | Matrix(_,e1,e2) -> if e1 != Noexpr || e2 != Noexpr then (
            check_int_literal_expr e1; check_int_literal_expr e2) else ();
             (t, id)
535       | _ -> (t,id)
536     in
537     {
538       styp = func.typ;
```

```
539         sfname = func.fname;
540         sformals = List.map check_formal_bind func.formals;
541         slocals = List.map check_var_decl func.locals;
542         sbody = List.map stmt_to_sstmt func.body;
543       }
544
545     in
546     let sfdecls = List.map fdecl_to_sfdecl functions in
547     (globals, sfdecls)
```

8.12 stdlib.px

```
:) Teresa Choe
:) Gabriel Kramer-Garcia
   def void printb(bool b){
    if(b){
        print_string("True");
    }
    else{
        print_string("False");
    }
}

def float[3][3] mat_int_to_float(int[3][3] a) {
  int i;
  int j;
  float[3][3] b;
  for(i = 0; i < 3; i = i+1) {
    for(j = 0; j < 3; j = j+1) {
      b[i][j] = $float a[i][j];
    }
  }
  return b;
}

def void print_veci(int[3] a) {
  int i;

  for(i = 0; i < 3; i = i+1) {
    print_int(a[i]);
  }

}

def void print_mati(int[3][3] a){
    int i;
    int j;
    for (i = 0; i<3; i=i+1){
        for (j = 0; j<3; j=j+1){
            print_int(a[i][j]);
            print_string(" ");
        }
        print_newline();
    }
}

def void print_matf(float[3][3] a){
    int i;
    int j;
    for (i = 0; i<3; i=i+1){
        for (j = 0; j<3; j=j+1){
            print_float(a[i][j]);
            print_string(" ");
        }
        print_newline();
    }
}

def int[3][3] mat_float_to_int(float[3][3] a) {
  int i;
  int j;
  int[3][3] b;

```

```
62    for(i = 0; i < 3; i = i+1) {
63      for(j = 0; j < 3; j = j+1) {
64        b[i][j] = $int a[i][j];
65      }
66    }
67    return b;
68  }
69
70  def float[3] vec_int_to_float(int[3] a) {
71    int i;
72    float[3] b;
73    for(i = 0; i < 3; i = i+1) {
74      b[i] = $float a[i];
75    }
76    return b;
77  }
78
79  def int[3] vec_float_to_int(float[3] a) {
80    int i;
81    int[3] b;
82    for(i = 0; i < 3; i = i+1) {
83      b[i] = $int a[i];
84    }
85    return b;
86  }
87
88  def int[3] scalar_mult_veci(int a, int[3] b) {
89    int i;
90    int[3] c;
91    for(i = 0; i < 3; i = i+1) {
92      c[i] = b[i] * a;
93    }
94    return c;
95  }
96
97  def float[3] scalar_mult_vecf(float a, float[3] b) {
98    int i;
99    float[3] c;
100   for(i = 0; i < 3; i = i+1) {
101     c[i] = b[i] * a;
102   }
103   return c;
104 }
105
106 def int[2][2] scalar_mult_mat2i(int a, int[2][2] b) {
107   int i;
108   int j;
109   int[2][2] c;
110   for(i = 0; i < 2; i = i+1) {
111     for(j = 0; j < 2; j = j+1) {
112       c[i][j] = b[i][j] * a;
113     }
114   }
115   return c;
116 }
117
118 def float[2][2] scalar_mult_mat2f(float a, float[2][2] b) {
119   int i;
120   int j;
121   float[2][2] c;
122   for(i = 0; i < 2; i = i+1) {
123     for(j = 0; j < 2; j = j+1) {
124       c[i][j] = b[i][j] * a;
```

```
125       }
126     }
127     return c;
128   }
129
130   def int[3][3] scalar_mult_mat3i(int a, int[3][3] b) {
131     int i;
132     int j;
133     int[3][3] c;
134     for(i = 0; i < 3; i = i+1) {
135       for(j = 0; j < 3; j = j+1) {
136         c[i][j] = b[i][j] * a;
137       }
138     }
139     return c;
140   }
141
142   def float[3][3] scalar_mult_mat3f(float a, float[3][3] b) {
143     int i;
144     int j;
145     float[3][3] c;
146     for(i = 0; i < 3; i = i+1) {
147       for(j = 0; j < 3; j = j+1) {
148         c[i][j] = b[i][j] * a;
149       }
150     }
151     return c;
152   }
153
154   def int vec_dot_producti(int[3] a, int[3] b) {
155     int sum;
156     int i;
157     sum = 0;
158     for(i = 0; i < 3; i=i+1) {
159       sum = sum + (a[i] * b[i]);
160     }
161     return sum;
162   }
163
164   def float vec_dot_productf(float[3] a, float[3] b) {
165     float sum;
166     int i;
167     sum = 0.0;
168     for(i = 0; i < 3; i=i+1) {
169       sum = sum + (a[i] * b[i]);
170     }
171
172     return sum;
173   }
174
175   def int[3] vec_vec_addi(int[3] a, int[3] b) {
176     int i;
177     int[3] c;
178     for(i = 0; i < 3; i = i+1) {
179       c[i] = a[i] + b[i];
180     }
181     return c;
182   }
183
184   def float[3] vec_vec_addf(float[3] a, float[3] b) {
185     int i;
186     float[3] c;
187     for(i = 0; i < 3; i = i+1) {
```

```
188     c[i] = a[i] + b[i];
189   }
190   return c;
191 }
192
193 def int[3][3] mat_mat_addi(int[3][3] a, int[3][3] b) {
194   int i;
195   int j;
196   int[3][3] c;
197   for(i = 0; i < 3; i = i+1) {
198     for(j = 0; j < 3; j = j+1) {
199       c[i][j] = a[i][j] + b[i][j];
200     }
201   }
202   return c;
203 }
204
205 def float[3][3] mat_mat_addf(float[3][3] a, float[3][3] b) {
206   int i;
207   int j;
208   float[3][3] c;
209   for(i = 0; i < 3; i = i+1) {
210     for(j = 0; j < 3; j = j+1) {
211       c[i][j] = a[i][j] + b[i][j];
212     }
213   }
214   return c;
215 }
216
217 def int[3] vec_vec_subi(int[3] a, int[3] b) {
218   int i;
219   int[3] c;
220   for(i = 0; i < 3; i = i+1) {
221     c[i] = a[i] - b[i];
222   }
223   return c;
224 }
225
226 def float[3] vec_vec_subf(float[3] a, float[3] b) {
227   int i;
228   float[3] c;
229   for(i = 0; i < 3; i = i+1) {
230     c[i] = a[i] - b[i];
231   }
232   return c;
233 }
234
235 def int[3][3] mat_mat_subi(int[3][3] a, int[3][3] b) {
236   int i;
237   int j;
238   int[3][3] c;
239   for(i = 0; i < 3; i = i+1) {
240     for(j = 0; j < 3; j = j+1) {
241       c[i][j] = a[i][j] - b[i][j];
242     }
243   }
244   return c;
245 }
246
247 def float[3][3] mat_mat_subf(float[3][3] a, float[3][3] b) {
248   int i;
249   int j;
250   float[3][3] c;
```

```
251   for(i = 0; i < 3; i = i+1) {
252     for(j = 0; j < 3; j = j+1) {
253       c[i][j] = a[i][j] - b[i][j];
254     }
255   }
256   return c;
257 }
258
259 def int[3] vec_mat_multi(int[3] a, int[3][3] b) {
260   int i;
261   int j;
262   int[3] c;
263   for(i = 0; i < 3; i = i+1) {
264     for(j = 0; j < 3; j = j+1) {
265       c[i] = c[i] + a[j] * b[j][i];
266     }
267   }
268   return c;
269 }
270
271 def float[3] vec_mat_multf(float[3] a, float[3][3] b) {
272   int i;
273   int j;
274   float[3] c;
275   for(i = 0; i < 3; i = i+1) {
276     for(j = 0; j < 3; j = j+1) {
277       c[i] = c[i] + a[j] * b[j][i];
278     }
279   }
280   return c;
281 }
282
283 def int[3] mat_vec_multi(int[3][3] a, int[3] b) {
284   int i;
285   int j;
286   int[3] c;
287   for(i = 0; i < 3; i = i+1) {
288     for(j = 0; j < 3; j = j+1) {
289       c[i] = c[i] + a[i][j] * b[j];
290     }
291   }
292   return c;
293 }
294
295 def float[3] mat_vec_multf(float[3][3] a, float[3] b) {
296   int i;
297   int j;
298   float[3] c;
299   for(i = 0; i < 3; i = i+1) {
300     for(j = 0; j < 3; j = j+1) {
301       c[i] = c[i] + a[i][j] * b[j];
302     }
303   }
304   return c;
305 }
306
307 def int[3][3] mat_mat_multi(int[3][3] a, int[3][3] b) {
308   int i;
309   int j;
310   int[3][3] c;
311   for(i = 0; i < 3; i = i+1) {
312     for(j = 0; j < 3; j = j+1) {
313       c[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j
```

```
            ];
314     }
315   }
316   return c;
317 }
318
319 def float[3][3] mat_mat_multf(float[3][3] a, float[3][3] b) {
320   int i;
321   int j;
322   float[3][3] c;
323   for(i = 0; i < 3; i = i+1) {
324     for(j = 0; j < 3; j = j+1) {
325       c[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j
            ];
326     }
327   }
328   return c;
329 }
330
331 def int[3][3] mat_transposei(int[3][3] a) {
332         int i;
333         int j;
334         int[3][3] b;
335         for(i = 0; i < 3; i = i+1) {
336             for(j = 0; j < 3; j = j+1) {
337                 b[j][i] = a[i][j];
338             }
339         }
340         return b;
341 }
342
343 def float[3][3] mat_transposef(float[3][3] a) {
344         int i;
345         int j;
346         float[3][3] b;
347         for(i = 0; i < 3; i = i+1) {
348             for(j = 0; j < 3; j = j+1) {
349                 b[j][i] = a[i][j];
350             }
351         }
352         return b;
353 }
354
355 def float det_mat2(float[2][2] a) {
356         return a[0][0] * a[1][1] - a[1][0] * a[0][1];
357 }
358
359 def float det_mat3(float[3][3] a) {
360         float[2][2] b;
361         float[2][2] c;
362         float[2][2] d;
363
364         b[0][0] = a[1][1];
365         b[0][1] = a[1][2];
366         b[1][0] = a[2][1];
367         b[1][1] = a[2][2];
368
369         c[0][0] = a[1][0];
370         c[0][1] = a[1][2];
371         c[1][0] = a[2][0];
372         c[1][1] = a[2][2];
373
374         d[0][0] = a[1][0];
```

```
375         d[0][1] = a[1][1];
376         d[1][0] = a[2][0];
377         d[1][1] = a[2][1];
378
379         return a[0][0] * det_mat2(b) + a[0][1] * det_mat2(c) + a[0][2] *
                det_mat2(d);
380 }
381
382 def float[2][2] mat_inverse2(float[2][2] a) {
383     float[2][2] b;
384     b[0][0] = a[1][1];
385     b[0][1] = -a[0][1];
386     b[0][0] = -a[1][0];
387     b[0][0] = a[0][0];
388         return (1 / det_mat2(a)) * b;
389 }
390
391 def float[3][3] mat_inverse3(float[3][3] a) {
392         float[2][2] b11;
393         float[2][2] b12;
394         float[2][2] b13;
395         float[2][2] b21;
396         float[2][2] b22;
397         float[2][2] b23;
398         float[2][2] b31;
399         float[2][2] b32;
400         float[2][2] b33;
401         float[3][3] c;
402
403         b11[0][0] = a[1][1];
404         b11[0][1] = a[1][2];
405         b11[1][0] = a[2][1];
406         b11[1][1] = a[2][2];
407
408         b12[0][0] = a[0][2];
409         b12[0][1] = a[0][1];
410         b12[1][0] = a[2][2];
411         b12[1][1] = a[2][1];
412
413         b13[0][0] = a[0][1];
414         b13[0][1] = a[0][2];
415         b13[1][0] = a[1][1];
416         b13[1][1] = a[1][2];
417
418
419         b21[0][0] = a[1][2];
420         b21[0][1] = a[1][0];
421   b21[1][0] = a[2][2];
422   b21[1][1] = a[2][0];
423
424         b22[0][0] = a[0][0];
425         b22[0][1] = a[0][2];
426         b22[1][0] = a[2][0];
427         b22[1][1] = a[2][2];
428
429         b23[0][0] = a[0][2];
430         b23[0][1] = a[0][0];
431         b23[1][0] = a[1][2];
432         b23[1][1] = a[1][0];
433
434         b31[0][0] = a[1][0];
435         b31[0][1] = a[1][1];
436         b31[1][0] = a[2][0];
```

```
437         b31[1][1] = a[2][1];
438
439         b32[0][0] = a[0][1];
440         b32[0][1] = a[0][0];
441         b32[1][0] = a[2][1];
442         b32[1][1] = a[2][0];
443
444         b33[0][0] = a[0][0];
445         b33[0][1] = a[0][1];
446         b33[1][0] = a[1][0];
447         b33[1][1] = a[1][1];
448
449         c[0][0] = det_mat2(b11);
450         c[0][1] = det_mat2(b12);
451         c[0][2] = det_mat2(b13);
452
453         c[1][0] = det_mat2(b21);
454         c[1][1] = det_mat2(b22);
455         c[1][2] = det_mat2(b23);
456
457         c[2][0] = det_mat2(b31);
458         c[2][1] = det_mat2(b32);
459         c[2][2] = det_mat2(b33);
460
461         return (1 / det_mat3(a)) * c;
462 }
463
464 def int[3] get_mat_rowi(int[3][3] a, int b) {
465   int[3] ret;
466   int i;
467   for(i = 0; i < 3; i = i+1) {
468     ret[i] = a[b][i];
469   }
470   return ret;
471 }
472
473 def float[3] get_mat_rowf(float[3][3] a, int b) {
474   float[3] ret;
475   int i;
476   for(i = 0; i < 3; i = i+1) {
477     ret[i] = a[b][i];
478   }
479   return ret;
480 }
481
482 def int[3] get_mat_coli(int[3][3] a, int b) {
483   int[3] ret;
484   int i;
485   for(i = 0; i < 3; i = i+1) {
486     ret[i] = a[i][b];
487   }
488   return ret;
489 }
490
491 def float[3] get_mat_colf(float[3][3] a, int b) {
492   float[3] ret;
493   int i;
494   for(i = 0; i < 3; i = i+1) {
495     ret[i] = a[i][b];
496   }
497   return ret;
498 }
```