## Contents

1 Introduction 3

2 Language Tutorial 3
  2.1 Statements and Variable declaration ........................................... 3
  2.2 Functions . ........................................................................... 3
  2.3 Main method . ....................................................................... 3
  2.4 Vectors . ............................................................................. 3
  2.5 Matrices .............................................................................. 4

3 Language Reference Manual 4
  3.1 Introduction . ......................................................................... 4
  3.2 Lexical Conventions . ............................................................... 4
    3.2.1 Comments . ..................................................................... 4
    3.2.2 Identifiers . ..................................................................... 4
    3.2.3 Keywords . ...................................................................... 4
    3.2.4 Literals . ......................................................................... 4
    3.2.5 Delimiters . ..................................................................... 5
  3.3 Syntax and Semantics . ............................................................... 5
    3.3.1 Type specifiers . ............................................................... 5
    3.3.2 Basic types . .................................................................... 5
    3.3.3 Vector . ........................................................................... 6
    3.3.4 Matrix . ......................................................................... 7
    3.3.5 Type conversions . ............................................................ 7
    3.3.6 Operators . ...................................................................... 7
    3.3.7 Matrix/vector operations . ................................................ 8
    3.3.8 Assignment . ................................................................... 9
    3.3.9 Functions . ..................................................................... 9
    3.3.10 Names and scope . ........................................................... 9
    3.3.11 Built-in Library . .............................................................. 9
  3.4 Statements . .......................................................................... 11
    3.4.1 If.. else . ...................................................................... 11
    3.4.2 Loops . .......................................................................... 11
    3.4.3 Blocks . ......................................................................... 12
    3.4.4 Return . ......................................................................... 12
  3.5 Formal Grammar . .................................................................. 12
  3.6 Sample Program . .................................................................. 14

4 Project Plan 16
  4.1 Process . .............................................................................. 16
    4.1.1 Planning . ...................................................................... 16
    4.1.2 Testing . ......................................................................... 16
  4.2 Style Guide . ......................................................................... 17
  4.3 Project Timeline . ................................................................... 17
  4.4 Roles and Responsibility . ........................................................ 17
  4.5 Development Environment . ................................................... 17
  4.6 Project Log . ......................................................................... 17

5 Architectural Design 42
  5.1 Diagram . ............................................................................. 42
  5.2 Scanner . ............................................................................... 42
  5.3 Parser . ................................................................................ 42
  5.4 Semantic Checking . ............................................................... 42
  5.5 Code Generation . ................................................................. 42

6 Test Plan 43
  6.1 Source Programs . ................................................................. 43
    6.1.1 Sample Program 1 . ......................................................... 43
    6.1.2 Sample Program 2 . ......................................................... 43
    6.1.3 Sample Program 3 . ......................................................... 44
  6.2 Test Suite . .......................................................................... 45
    6.2.1 Test Suite Code ............................................................... 45
6.2.2 Output of running test suite ........................................ 48
6.2.3 How Test Cases Were Chosen ...................................... 52
6.2.4 Automation Used While Testing .................................... 52
6.3 Responsibilities ....................................................... 52

7 Lessons Learned ......................................................... 52

8 Appendix ................................................................. 53
  8.1 ast.ml ................................................................. 53
  8.2 codegen.ml ......................................................... 56
  8.3 inputPic.c ........................................................... 63
  8.4 Makefile .............................................................. 65
  8.5 makePic.c ............................................................ 67
  8.6 parser.mly ............................................................ 68
  8.7 pixelman.ml .......................................................... 72
  8.8 printbig.c ............................................................ 73
  8.9 sast.ml ............................................................... 75
  8.10 scanner.mll ......................................................... 76
  8.11 semant.ml ........................................................... 78
  8.12 stdlib.px ............................................................ 90
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```java
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:

<table>
<thead>
<tr>
<th>void</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>return</td>
<td>bool</td>
</tr>
<tr>
<td>def</td>
<td>false</td>
</tr>
<tr>
<td>main</td>
<td>if</td>
</tr>
<tr>
<td>else</td>
<td>Vector</td>
</tr>
<tr>
<td>int</td>
<td>for</td>
</tr>
<tr>
<td>float</td>
<td>while</td>
</tr>
<tr>
<td>Matrix</td>
<td></td>
</tr>
</tbody>
</table>

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.

4
3.2.4.3 Boolean literals  The boolean literal \texttt{true} is stored as a byte of value 1. The boolean literal \texttt{false} is stored as a byte of value 0.

3.2.4.4 Character literals  A character literal is a single character surrounded by single quotes \texttt{	extquoteleft} \textquoteleft 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.

\[
\begin{array}{c}
\text{int[2][2] a;}
\end{array}
\]

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.

\[
\begin{array}{c}
\text{int x = 3; :) an integer of 3}
\text{def int myFunction(){} :) return an integer}
\end{array}
\]

3.3.2 Basic types

3.3.2.1 Integers  Pixelman supports integers through the \texttt{int} type. An example of declaring and initializing an \texttt{int} variable:

\[
\begin{array}{c}
\text{int i;}
\text{i = 42;}
\end{array}
\]
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:

```c
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:

```c
bool isColor;
bool isNonzero;
isColor = true;
isNonzero = 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.

```c
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.

```c
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.

```c
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

```c
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 be 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:

```c
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.

```cpp
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.

```cpp
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.

```cpp
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.

```cpp
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.

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

3.3.7 Matrix/vector operations The operations +, -, and * are supported for vectors 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.
  
  \(- A \times B \) will return a 3 by 3 matrix which is the matrix product of the two matrices.

• If one operand is a \textit{int} or a \textit{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 \textit{float} and one is of type \textit{float} or type \textit{int}, the return type will be of type \textit{float}.

• If both operands are of type \textit{int}, then the return type will be of type \textit{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:

\[
*=\ /\ =\ %=\ +=\ -=\ \texttt{\&}=\ ^|=\ ]=\]

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:

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

where \texttt{\<type\>} is the return type of the function, \texttt{\<functionName\>} is the unique label for a function, and \texttt{\<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:

```python
\texttt{\<functionName\>(\texttt{\<arg1\>...\<argn\>);}\n```

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 column 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.

```python
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.

```python
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.

```python
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 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:

```python
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**

   ```plaintext
   program: decls EOF
   ```

2. **Declarations**

   ```plaintext
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 ;

ev_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 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 ]
  [1 matrix_literal ]
The primary-expression operators ( ) [ ] have highest priority and group left-to-right. The unary operators ! - (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.

```
assignop:
  =
```

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.

```python
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("a[0][0] = ");
  print_newline();
  print_int(a[0][0]);
  print_newline();

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

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

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.

<table>
<thead>
<tr>
<th>Team Member</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony Chan</td>
<td>Compiler Frontend, Semantic checking, Debugging, stdlib</td>
</tr>
<tr>
<td>Brian Tsau</td>
<td>Compiler Backend, Semantic checking (&amp; SAST)</td>
</tr>
<tr>
<td>Gabriel Kramer-Garcia</td>
<td>Compiler Frontend, Linking, C Library, Test Suite</td>
</tr>
<tr>
<td>Teresa Choe</td>
<td>Compiler Frontend, Compiler Backend, Code Generation, stdlib</td>
</tr>
</tbody>
</table>

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 Ocamlex 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

```
1 commit 6281a22a1be6c9351564448c4bfc33761d8413aa
2 Author: anthony chan <tn.chan5@gmail.com>
3 Date: Wed Dec 20 16:21:43 2017 -0500
4
5    fix stdlib. FINAL
6
7 commit 0849b4164137a15bda1172e00e1ebe1d140a2859
8 Author: acbeast <3113077+acbeast@users.noreply.github.com>
9 Date: Wed Dec 20 15:55:48 2017 -0500
10
11 add spaces to print_vecf/i
12
13 commit e52fc79683b291933e500f030b13e6dd105723b
14 Merge: 484f7af e02d829
15 Author: anthony chan <tn.chan5@gmail.com>
```
Merge branch 'master' of https://github.com/btsaubt/pixelman

commit e02d8291c03efc6e421ff4dd2ff76c2b7995b6
Merge: 1e254d3 596f654
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 20 13:13:55 2017 -0500

merge

commit 484f7af89c9a405bdcd3abfad359fe9af67de0c3
Merge: 03f7460 1e254d3
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 20 13:13:32 2017 -0500

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

commit 03f7460f144655a4c53e576ae378ff2b714611a0
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 20 13:12:54 2017 -0500

fix vector.ppx demo file (add newlines), remove primitives.ppx

commit 596f65493b6db4aa148d7df7bc61f60c50e39c9e
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 20 13:12:44 2017 -0500

pretty printing

commit 1e254d3a213ef092cd46460e57a31dc0972a04c2
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Wed Dec 20 13:11:41 2017 -0500
tests

commit 23e23653c2e2fc4a6ab8b74b607b9b616e3d917e
Merge: d510d766f8b903
Author: btsaubt <btsaubt>
Date: Wed Dec 20 13:00:14 2017 -0500

Merge branch 'matrices'

commit df8b903e2ba7f38bcedb96ad09202e35d26ea2b35
Author: btsaubt <btsaubt>
Date: Wed Dec 20 13:00:06 2017 -0500

added inverse * a (doesn't work right)

fix tests

commit 921333c060621a75ef6d35c39e1f3e48cb02848
Merge: 32383b1 1c87de3
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 20 12:50:06 2017 -0500

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

commit 32383b175cdc94991df3c589769b8b7aa1ab013c
fix all warnings and comment out image

commit 1c87de3e86a1a9f0f61eccc058e22342912c4c14
Merge: 31edfe9 414a063
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 20 12:46:09 2017 -0500
merge

commit 31edfe9fc9611e27b257d7221cd84970fd6c56c
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 confl

commit f1a724cbd596d463d526356057bd1db72c13377d
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 20 11:41:23 2017 -0500
changes to std

commit 19a62c07db92a9875f8e8e566a40459b1e6d624a
Merge: f8cc077 921333e
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 <gkl2110@columbia.edu>
Date: Wed Dec 20 12:42:49 2017 -0500
add library function

commit b4dadd7b3a3f26d19e84e6dbdae54945cbf07cc
Author: btsaubt <btsaubt>
Date: Wed Dec 20 11:29:19 2017 -0500
added print_newline()
commit 4afcd5a77bb06b725aaad77889de8a64651042e
Merge: abb3e8 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 70f769167da8ac89b41326ec6a3aadcedeb5dda5
Author: btsaubt <bt2420@columbia.edu>
Date: Wed Dec 20 12:18:19 2017 -0500

got inverse for 3x3

commit 1c1515bb6a38e8bb3ea547935157655bb89130d
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 e45afcacf2babc31e547b9aa701d7f6e5de875423
Merge: 5e00f4b 3990091
Author: btsaubt <btsaubt>
Date: Wed Dec 20 02:03:04 2017 -0500

Merge branch 'master' into matrices

commit 5e00f4b60f6c1f16f1247776dc2bcac7b9869ab4
Author: btsaubt <btsaubt>
Date: Wed Dec 20 02:02:43 2017 -0500

added det and transpose to stdlib

commit 399009138e38d24e783233362524ae714d3ed649d
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 20 01:28:53 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 e31accf44b14bffeeee869b80e46802e9972add5b
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 20 01:18:11 2017 -0500

sample program skeleton

commit ac87fb3abf6de8dc3eb83ce8efde8a3646f73c1b
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Wed Dec 20 01:14:05 2017 -0500

commits

commit f38bd18d63f4ad80b7cd06f11c19cadfa8bcc63b
Merge: 9cf1fadb eb60756
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Wed Dec 20 01:11:02 2017 -0500
Merge branch 'master' of https://github.com/btsaubt/pixelman

commit 9cf1fadbec043af5bfe78a4285566fa597cbf233
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Wed Dec 20 01:10:54 2017 -0500
add input output methods

commit eb60756e1b28c70be2dab4abf73cbda3cb4f4e8
Merge: 67f5164 7da8504
Author: btsaubt <btsaubt>
Date: Tue Dec 19 22:28:25 2017 -0500
Merge branch 'master' into matrices

commit 67f516470b948573e8c92d69976774a539d1
Author: btsaubt <btsaubt>
Date: Tue Dec 19 22:28:09 2017 -0500
matrix/vector library functions added, sum typos fixed

commit 7da85044df8e7f7704d8fa2520b29f23067741
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Tue Dec 19 22:15:19 2017 -0500
fix tests

commit fd040b0ddde97e8d18ad1dec4d184661cd721993e
Merge: 0c29611f b548011
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Tue Dec 19 22:14:50 2017 -0500
Merge branch 'master' of https://github.com/btsaubt/pixelman

commit b5480113a1a154f23fdefa8d48fb685cf8c59076bb
Author: anthony chan <tn.chan5@gmail.com>
Date: Tue Dec 19 22:14:23 2017 -0500
image access semantics checking works

commit 0c29611f918fdefaf745660577378aea084aba35b2
Merge: f17b3fe 48f8936
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Tue Dec 19 22:11:17 2017 -0500
Merge branch 'master' into gabe

commit f17b3fe6e4136e804e7a53f71b19d5e64185c0853
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date: Tue Dec 19 22:09:20 2017 -0500
makePic

merge conflict
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 0f1fa2e2e759ca0b90f8ae15d7d8dd51d413529dfa
Merge: 4a21cffe ed9e1f1
Author: anthony chan <tn.chan5@gmail.com>
Date: Tue Dec 19 21:49:31 2017 -0500

image access parse, ast, sast

commit ed9e1f1e23ac3ec592b923a57faceddda87d2fe
Author: anthony chan <tn.chan5@gmail.com>
Date: Tue Dec 19 21:46:08 2017 -0500

image access parse, ast, sast

commit 4a21cffe1974c38b5c80c320a9a7358a8c53f1cb5
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 b8bdcc230a481738465ac6cab99b812e2adb270
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 efe8aa7092075d1d3a2b85d3afdf2734b6ae7d7f1
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
Author: btsaubt <btsaubt>
Date:  Tue Dec 19 00:30:50 2017 -0500

Merge branch 'master' into matrices

commit 0bcc2d0fd7e571ee347ea5b1fe8f6dae64d108ac
Merge: f5da6df 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 30e5d308b0b19339f39ab59f12b5202eb9b3930ac
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date:  Tue Dec 19 00:08:48 2017 -0500

changes

commit 5103939296b62adcdbe1e2748d96d5503570b9861
Author: btsaubt <btsaubt>
Date:  Mon Dec 18 23:25:48 2017 -0500

got rid of warnings

commit a1206c31b532d56a10eaf6abcde0e25a259b3049
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 34d5bd077579c02593b0db551c3d17c8ed4baac7
commit 4e27d827d90c629c06bc33cbad66eef147740a3b
Author: btsaubt <bt2420@columbia.edu>
Date:  Mon Dec 18 21:54:05 2017 -0500

  test for vector addition

commit 9400e59d53155e72e37d6f3902e781de962baee6
Author: Anthony Chan <tn.chan5@gmail.com>
Date:  Mon Dec 18 20:30:29 2017 -0500

  work on operator overloading

commit 3619d13336e96c234ed4e33f5bac74bc0baf8ce
Merge: f2e1ca8 8277738
Author: btsaubt <bt2420@columbia.edu>
Date:  Mon Dec 18 14:29:46 2017 -0500

  attempt at vector/matrix length

commit 82777382dce8b3f0904a1f066d1fd0ba4f006f5f
Author: Teresa Choe <tc2716@columbia.edu>
Date:  Mon Dec 18 14:29:07 2017 -0500

  added int -> float type inferencing in assignment and function actuals

commit 2ab4d8131762909628efbdad4021e4b4e000a71
Merge: 44f6ea8 642590a
Author: Teresa Choe <tc2716@columbia.edu>
Date:  Mon Dec 18 13:33:31 2017 -0500

  merge conflicts

commit 642590ae4496294c7296bebecd7f0e5df0c46a31
Author: Gabriel Kramer–Garcia <glk2110@columbia.edu>
Date:  Mon Dec 18 13:32:36 2017 -0500

  add loops back to tests

commit 5cb8d769e0425768f9b90e6d939a3e12e8b3c283
Author: btsaubt <bt2420@columbia.edu>
Date:  Mon Dec 18 13:26:15 2017 -0500

  fixed basic block for matrix/vec access

commit 44f6ea87344d4e0b4188a5b454f0ee634de7af7
Merge: cb6f379 bb7cc46
Author: Teresa Choe <tc2716@columbia.edu>
Date:  Mon Dec 18 13:22:19 2017 -0500

  merge conflicts
commit cb6f379eeef12ccee139d6501ba756c5b18b62a37
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 d7276e0305e66f1f6550326b4385a52929ebe7da9
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 926b3f43b79068c8b7c4ab92cf67a8aa5605aa91
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 d57cda82888c3085a01368be7ee2ab86e696f9e0
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Dec 18 13:14:45 2017 -0500
fix failing tests

commit ecbf909be8ae7de2dc843b17e9d2c8e82c45d1bfc
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Dec 18 12:51:20 2017 -0500
fix cast tests

commit f30d29f6695e58c30d6de90e20d9ce843f0e81fb
Merge: c86c437 2d36ec3
Author: btsaubt <bt2420@columbia.edu>
Date: Mon Dec 18 01:24:00 2017 -0500
merge conflicts

commit c86c43761f1a55ebbe6b523edf7d5bd27ce6156e
Author: Teresa Choe <tc2716@columbia.edu>
Date: Mon Dec 18 01:23:31 2017 -0500
added multiline comments yay

commit 2d36ec3c5cd08fa07a404ad0079e9ac9536bdf
Merge: ffe431c f9e9b5a
Author: btsaubt <btsaubt>
Date: Mon Dec 18 01:22:06 2017 -0500
fixed merge conflict

commit ffe431c84bfae996329ba02c29663350b1fd8c7
Author: btsaubt <btsaubt>
Date: Mon Dec 18 01:15:38 2017 -0500
added pointers to grammar, started matrix operator overloading

commit f9e9b5a953d747b7d3b36ac6dd509e3707250c6a
Author: Teresa Choe <tc2716@columbia.edu>
Date: Mon Dec 18 01:08:31 2017 -0500

added matrix literal to codegen

commit 6fd84e5f94f104ac1932f252381eaaece01bc67d
Author: btsaubt <btsaubt>
Date: Sun Dec 17 23:32:08 2017 -0500

added vectors/matrices to types

commit 2e2b4e3373901b0d9fa1e230e8a8b5ada8bc46d0
Author: btsaubt <btsaubt>
Date: Sun Dec 17 22:39:19 2017 -0500

tests for vector element access and assignment

commit 5ffda4a99f1a3cfcab3af67293dbd138efa603
Merge: 76cc926 06377b4
Author: btsaubt <btsaubt>
Date: Sun Dec 17 22:35:39 2017 -0500

fixed merge conflicts

commit 76cc926dbf99e73d1ee2625e5d107c4434bafcfb
Author: btsaubt <btsaubt>
Date: Sun Dec 17 22:34:46 2017 -0500

got vector/matrix access to make!

commit 06377b43fb5af810cb51243ba603be15e403f2f8
Merge: a9550e2 471dcff
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Dec 17 20:27:32 2017 -0500

merge conflict

commit a9550e2731469aad1ea7b542f39983be04eeb4f
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Dec 17 20:27:10 2017 -0500

Added bitwise operators.

commit 471dcffba7e71398f716700241772465b7750e5e
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 17 19:50:43 2017 -0500

added cast tests

commit 699bcece7ee28c30476df062d82bed301a95e172b6
Merge: 790fa6f 810cb29
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 17 19:43:50 2017 -0500

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

commit 810cb2974740930d3187fc72f54f1977d032b82
Author: btsaubt <btsaubt>
Date: Sun Dec 17 19:42:52 2017 -0500
removed stuff from stdlib

commit 790fa6f382c0fed119876ce976b2858257220cf6
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 17 19:35:35 2017 -0500

more tests

commit 1c54ffaa984b3c875b1548ce13095f27bd4b3274
Author: btsaubt <btsaubt>
Date: Sun Dec 17 19:33:51 2017 -0500

moved parser back

commit 85846a530235120b8decda7ed2aabaaf20011584
Merge: 9961b9e5c059a2
Author: btsaubt <btsaubt>
Date: Sun Dec 17 19:30:27 2017 -0500

Merge branch 'master' into matrices

commit 9961b9e391a52279909e620c60f2d88d54dd7a8
Merge: 7ab547a99e2129
Author: btsaubt <btsaubt>
Date: Sun Dec 17 19:29:37 2017 -0500

fixed merge conflict

commit 5c059a2868a878e8d2781733634dcd8ee56a4aa
Merge: 0c49ab5 99e2129
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Dec 17 19:26:47 2017 -0500

merge conflict

commit 0c49ab568b71c2ace41cfb6952c9ae2dd6ee4b9
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Dec 17 19:26:27 2017 -0500

vector literal in code gen and added test

commit 7ab547ab3e291742113842709de38b938d3af235
Author: btsaubt <btsaubt>
Date: Sun Dec 17 19:23:15 2017 -0500

fixed shift/reduce for casting

commit 99e2129d4eeae978e24e22f537657ae1b2d38e89
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 17 19:16:20 2017 -0500

fix matrix tests

commit b1bc802a41509d19693d29b9dc40d08907c88ee9
Merge: 554ba68 09b7520
Author: anthony chan <tn.chan5@gmail.com>
Date: Sun Dec 17 19:11:32 2017 -0500

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

commit 554ba68b1508683a794dc1b97a7ef9a312322ac
Author: anthony chan <tn.chan5@gmail.com>
Matrix semant works

commit 09b75200a1f35ab34858283ef3359e6239cd7d8
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 17 19:10:41 2017 -0500

changed print to print int and corresponding tests

commit 809be70dc523c204d3139e044009e50cdabb1680
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 17 18:57:05 2017 -0500

another test

commit ab057e02c3169188fafa4c759081605c99715855
Merge: fb3dfe4f9878f7
Author: btsaubt <btsaubt>
Date: Sun Dec 17 18:52:30 2017 -0500

Merge branch 'master' into matrices

commit fb3dfe4a80da53c96cd466a9a11197b730acc1420
Merge: btsaubt <btsaubt>
Author: anthony chan <tn.chan5@gmail.com>
Date: Sun Dec 17 18:49:40 2017 -0500

added type casting to grammar and semant

commit 1130c7448fedc5198730050917509eac0da075f9
Merge: 075ad6bf9878f7
Author: anthony chan <tn.chan5@gmail.com>
Date: Sun Dec 17 18:49:17 2017 -0500

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

commit 075ad6b7a5b204018a2be41af6f76e377ada1668
Merge: 396839c3246a7d
Author: anthony chan <tn.chan5@gmail.com>
Date: Sun Dec 17 18:40:37 2017 -0500

Merge branch 'vector_semant'

commit 396839c6b5515d1bc39d4286168cfff1561a1cf89
Merge: 2927c8e0de0a27d
Author: anthony chan <tn.chan5@gmail.com>
Date: Sun Dec 17 18:40:20 2017 -0500

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

commit 3246a7d66e7d787982cc398e72134cde33b268fa
Merge: 2927c8e0de0a27d
Author: anthony chan <tn.chan5@gmail.com>
Date: Sun Dec 17 18:40:02 2017 -0500

Vector semant checking functional

commit f9878f72ccfe99d0a9b690c2947092910e3dda1b
Merge: c771ae0de0a27d
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Dec 17 18:38:11 2017 -0500

merge conflict
commit c771ae0b0f226966e926788de8674791a0a51ec
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Dec 17 18:37:19 2017 -0500

    Changed grammar of matrices

commit de0a27db14c3b1f2c55e63916c732bcd7475b99
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 2927e8e76ae802ccbe937d950e7ab4c8db9569f95
Merge: 2d4bacbf81000a
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 f81000a65a6c12efef1b9d044503741267f427d9a
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 d2d481751648ae136626e87f97e425f74ead3020a
Author: btsaubt <btsaubt>
Date: Sat Dec 16 23:53:37 2017 -0500

    commented out stdlib functions, changed make clean to remove err

commit 3f2f8b4426475e2b00af7b9460308608fa1a943
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 c2e6430
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 c2c64306elee2703121ff91f4874662750ecece96
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sat Dec 16 16:05:38 2017 -0500

add grayscale

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

added printb

commit 35d2b9558c331c5d8ecab9638212794632cd91fc
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sat Dec 16 14:45:59 2017 -0500

added print_string to protexed functions

commit 3fd072dfe96b6dd156ac0cd6354274e1ae9a18f6be
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sat Dec 16 14:56:57 2017 -0500

Merge: be7b274c067c7cfb2ddd0a82317e153e2888a6ec30a

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

fix unary and binary minus differentiation

commit be7b274c067c7cfb2ddd0a82317e153e2888a6ec30a
Author: Anthony Chan <tn.chan5@gmail.com>
Date: Sat Dec 16 00:02:28 2017 -0500

fixed merge conflicts

commit 067c7cfb2ddd0a82317e153e2888a6ec30a
Author: Anthony Chan <tn.chan5@gmail.com>
Date: Sat Dec 16 12:21:16 2017 -0500

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

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

fixed merge conflict

Commit fda39766eb1bab5a00a0f6f208b180aff00a1c75
Author: btsaubt <btsaubt>
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
Date: Fri Dec 15 23:04:35 2017 -0500

added vec/mat init in codegen (not tested yet), and started vec/mat
literals in semant

commit 455b8dd469616d1b2db1e4a03cc8ed0355c8317
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Fri Dec 15 17:58:57 2017 -0500

connected stdlibgit status git status and wrote printb

commit e46cc436b2b124d6573016584053b89391f85734
Merge: 95d40c4 a06c926
Author: btsaubt <btsaubt>
Date: Fri Dec 15 16:28:24 2017 -0500

Merge branch 'matrices'

commit a06c926e5e7f007e28d6aa3c2b6979165fcdbec4
Author: btsaubt <btsaubt>
Date: Fri Dec 15 16:28:01 2017 -0500

fixed errors caused by merge conflict

commit 95d40c4120affadee851b8ad087c8937f86778
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Fri Dec 15 16:19:50 2017 -0500

refix tests

commit 74df167a533403ae6ebd84436d22e72cf152bc6d
Merge: cle4a5f cae7c5a
Author: anthony chan <tn.chan5@gmail.com>
Date: Fri Dec 15 15:51:17 2017 -0500

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

commit cle4a5f9216dc56a39b01e98a4cc3531fcd4ed0c
Author: anthony chan <tn.chan5@gmail.com>
Date: Fri Dec 15 15:50:40 2017 -0500

Add printb back for printing bools

commit cae7c5a7831920098d454048cfd37d1775f280e9
Merge: e4db2bb 6da125f
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Fri Dec 15 15:44:31 2017 -0500

Merge branch 'stdlib'

commit 6da125fd2c9ca661d0761ce4a567f048a11e1cb4a
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Fri Dec 15 15:42:43 2017 -0500

test stdlib

commit 4eb2bb13e48dd472f6f4e9312d90a3c28d66ca2
Merge: cc00d0c bcd41e9
Author: Teresa Choe <tc2716@columbia.edu>
Date: Fri Dec 15 15:22:15 2017 -0500

merge conflicts
test to check vector and matrix literals

Matrix literals done

edit pixelman file

Change ast to sast and fix codegen for sast

added literals for vectors

added vector/matrix access in grammar and semant

added checking vector/matrix dimensions for int in semant

Merge branch 'master' into matrices
commit 3e6955f48be23487de6f855a644463fceeae1c411
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 a4a56ec6ca32bf6e3e7a4bbcedde0efca31
Author: btsaubt <btsaubt>
Date: Thu Dec 14 18:17:16 2017 -0500

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

commit 12eda893a72c98d516cd2a51696e824869b4145
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 18:09:01 2017 -0500

more tests

commit a1a5e0f1a2c55639fd6f108152dd6cb15092c6b
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 0c46c214ed69c128d3705e5c6e6ed94d5cdd77
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 17:42:07 2017 -0500

changed texts to work with printf_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 b5c6f257174fe2b1dd6b0eeca8398ee5e5118fb8c
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 16:37:52 2017 -0500

more tests

commit dab028578c90753c70e570d3a05dfb06780e2a
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 83d468e96a5a53fe769df2098ec59702697ee555de
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 e6f7d74db370651b06766fc96db51642e4d69a2
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 15:53:14 2017 -0500

fix test files

commit 85ea7d94a923bd635eaa3407d35d99acc0269ff6
Merge: 6d222d7 c15dc32
Author: Teresa Choe <tc2716@columbia.edu>
Date: Thu Dec 14 15:48:43 2017 -0500

fixed merge conflicts.

commit 6d222d7a14352ea7ed40ce0c095a8ec22fd3611
Author: Teresa Choe <tc2716@columbia.edu>
Date: Thu Dec 14 15:47:43 2017 -0500

fixed greater than for integers

commit e15dc328cc3915e15dbdb39f336f26fe920b9a77
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 15:46:10 2017 -0500

changed fail tests to .px

commit 13d5a282810119a9fl182fb57e4daa4bd8428e51
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 15:40:39 2017 -0500

fix tests

commit 8ed54db0ec8b08b656ba0ad3d055f2a17d105cf
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 15:07:22 2017 -0500

change to our test suite

commit 3d0bcc867294fe9d170088b4d5c616e10e10ac
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Thu Dec 14 14:16:42 2017 -0500
renamed everything

commit 7b3f2fca49a34d463aa5de3ce62014a225654c39
Author: btsaubt <bt2420@columbia.edu>
Date: Thu Dec 14 13:16:49 2017 -0500

  Added to semant, minor changes to all to generate sast from sast

commit 3f5301f1a519884a8451d0e53e72ee6653ad5899
Author: btsaubt <bt2420@columbia.edu>
Date: Wed Dec 13 22:33:08 2017 -0500

  changes to semant

commit 013deac94580d745e52e77f538e4e24f86cfded
Merge: 2e9f56e 132cc5a
Author: btsaubt <btsaubt>
Date: Wed Dec 13 21:00:06 2017 -0500

  Merge branch 'master' into matrices

commit 2c9f56e5666f767798e8b89242be98cd156112ec
Author: btsaubt <btsaubt>
Date: Wed Dec 13 20:59:42 2017 -0500

  added sast (not implemented yet)

commit 132cc5a05fc5c214b59382a42be7c98e1b184ad8
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 20:53:31 2017 -0500

  Forgot to add float test for ast, and it also compiles.

commit 523ba6f4e1d21d91c7046092525c59ab4dceeeea9
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 20:52:06 2017 -0500

  Fixed floating pt regex.

commit 9476b893490f9a3e5a4bd7f9ba0637adc3c4a57e
Merge: 63e7490 5813238
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 20:37:13 2017 -0500

  merge conflicts

commit 63e74905075c318a84e962819e2299b8bb7cdebf
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 20:35:49 2017 -0500

  Float operators, bitshift, and tests.

commit 58132383fed65411600be921504fe283a704c653
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 19:03:20 2017 -0500

update test

commit 35536cfe2bdab7a6d5a341eed5b7666ea7eee88
Merge: 193aebf 774770e
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 13 19:01:20 2017 -0500
Merge branch 'master' of https://github.com/btsaubt/pixelman

commit 193aebfc2699e3a7ec29024b086dc8ed33c73ec23890
Author: anthony chan <tn.chan5@gmail.com>
Date: Wed Dec 13 18:59:33 2017 -0500

  Fix print_string to not print quotes

commit 143e7213c5ce541e653b5484826a1841fb568d2a
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 17:48:23 2017 -0500

  Added shift right/left bitwise.

commit 774770efa020a188eeade9096d4025a71e59c59d0
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 17:43:35 2017 -0500

  fix test

commit b723b2a99af17e6553b323a2197df63518c0e3e6
Merge: 307084c 8913c1c
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 17:14:53 2017 -0500

  added modulus

commit 8913c1c1f86fda87eb1265c36212779ad63
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 17:04:01 2017 -0500

  back to ast

commit 7a91d3e087940bc59bfa4319594095e2b7305eeb
Merge: 5977da2 cd9f1b7ab
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 16:45:35 2017 -0500

  Merge branch 'master' into tests

commit cd79babf2af3b8a3a98959dfbe6b73b71e765
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 16:42:21 2017 -0500

  Fixed shift/reduce error

commit 5977da2f98b9c824dc6d4fdeae1641a77c89
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 16:39:29 2017 -0500

more tests

commit e713deed37a668fabe528ad1a91c6d393d4898a
Merge: 3bafdd4 81f78f
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 16:29:13 2017 -0500
Merge branch 'master' into tests

commit ecd7f34203ed0ee5549df6d1f518cda02bf1fe68
Merge: de0d70b 81fc78f
Author: btsaubt <btsaubt>
Date: Wed Dec 13 16:28:08 2017 -0500

fixed stupid merge conflict git status

commit 3bafdd400d2b5785afe32159e2a84b07f1fb9782e
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 16:24:57 2017 -0500

commit 3bafdd400d2b5785afe32159e2a84b07f1fb9782e
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Wed Dec 13 16:24:57 2017 -0500

did stuff

commit de0d70b2316db38cb8af56756ffbd58d92413eb
Author: btsaubt <btsaubt>
Date: Wed Dec 13 16:23:43 2017 -0500

commented out some stuff to implement matrices

commit 81fc78f5ee7a1da2a9523391a4822a21159ba186
Merge: 1020b4b e42f2e2
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 16:12:37 2017 -0500

All merge conflicts fixed

commit 1020b4b6cc0ff826dd35a19851a0f0a9414fe6434
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Dec 13 16:10:05 2017 -0500

operators, fixed tests, added to parser

commit ba09486dd63003077ef02a715db10a27ff41bae3
Author: btsaubt <btsaubt>
Date: Wed Dec 13 16:09:26 2017 -0500

changes to semant for matrices

commit e42f2e245f32e31905c4a92d60de258f5187fdab
Merge: 539b64c 335840c
Author: btsaubt <btsaubt>
Date: Tue Dec 12 12:19:02 2017 -0500

Merge branch 'matrices'

commit 335840ce0e42331eece667dfb461f690946d19cf
Author: btsaubt <btsaubt>
Date: Tue Dec 12 12:18:12 2017 -0500

Merge branch 'master' into matrices

commit c14b5d6c2bddd879c0da046e71b3ee01e33214e8
Author: btsaubt <btsaubt>
Date: Tue Dec 12 12:16:47 2017 -0500

fixed matrix in grammar

commit 539b64c84b464b6d3c6c002b233e772bb7abb2cc
Merge: 134f8b3 78b698f
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Tue Dec 12 12:15:55 2017 -0500
Merge branch 'master' into tests

commit 78b698f7475851b96f21e88fd34f78794f853d9b
Author: btsaubt <btsaubt>
Date: Tue Dec 12 11:52:12 2017 -0500

added matrices and vectors to grammar (access and initialization)

commit 134f8b37add1b612a409025519ac0609891f77aa
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Dec 10 20:47:22 2017 -0500

added tests

commit 63239ece5630712731d591f6c8d83fb0fa8a4efc
Author: Teresa Choe <tc2716@columbia.edu>
Date: Thu Dec 7 19:25:51 2017 -0500

Added types and bitwise to semant.

commit dff6fa813905f48353ead2b80a4e9b7305f97e27
Author: Teresa Choe <tc2716@columbia.edu>
Date: Thu Dec 7 19:20:41 2017 -0500

Added :) as our single-line comments.

commit 17462a8bff236bfd3a63039f5ac90c84346a836f
Merge: 50082c2 22fc109
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Nov 29 21:02:28 2017 -0500

Finished grammar

commit 50082c2b6ebb954886d1d2ab3c3a0ea7d81029ac
Author: Teresa Choe <tc2716@columbia.edu>
Date: Wed Nov 29 21:00:56 2017 -0500

Finished most of grammar

commit 22fc1097f7a19456d30d335aca3430972d94d86f
Merge: f959300 1a2af77
Author: btsaubt <bt2420@columbia.edu>
Date: Wed Nov 29 20:32:32 2017 -0500

Merge branch 'master' into tc2716

commit 1a2af77659e7dac9fb94c5f17ee90968e45bca2d
Merge: 52e678d 683fe9b
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Nov 19 22:55:12 2017 -0500

Merging branches

commit 52e678dc3eac801edcc1a3bd904216e7a85b6ae5
Merge: 09a9d43 1b09c78
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Nov 19 16:57:20 2017 -0500

Changes to scanner.

commit f959300aa3f8ffbc3b0ac100aae3a73773897de2
Merge: 683fe9b 2bb1bb4
Merge branch 'tc2716' of https://github.com/btsaubt/pixelman into tc2716

Merge master

Commit 2bb1bb4355bcbcbcc8a802b0e34ff79cab126ac5bb
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 9777dd890907c9d3af0b64dc914cc35a4e7a9092
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Nov 19 18:36:02 2017 -0500

Added bitshift and float to ast and parser.

Commit 683fc9b17666589d1561797da6698e93d18be55d
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 b6425b89fe9c727b3f79f9a9f3c4ba8e3588caa
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 6524d2d43f4955867f8ba9ce1e376d6dc63150f
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Nov 13 21:39:17 2017 -0500

Hello world

Commit ea4fa7a19ca4c997556fec290b667d10dea36dd
Merge: 948d148 1b09c78
Author: Teresa Choe <tc2716@columbia.edu>
Date: Sun Nov 12 18:38:37 2017 -0500

Updated scanner

Commit 1b09c788c2698787895e5d1963c55cf7ec4693d0
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 4fabfa54e28ec7352e65bbd2a77aabe569b200da
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 cb3d38e3cf9d448ea1a7fb831224b9bff2adeff7
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 45300b28f403a8dd3ffa478f5b7293f0d36a5e7
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 64ff70b614c3c69ae6611bcb7a6dc89e05db
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
commit 2698cf216cbfe642ddfa4f906f967cac0f0cd0b3
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Nov 6 15:34:58 2017 -0500
fix Literal

commit 1fa6108d0f978f37a046111757bb43aea43900b46
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Nov 6 15:33:30 2017 -0500
more changes

commit 4938580bb5fac381578a7985e6d75a72bd9d6b8b
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Nov 6 15:29:20 2017 -0500
removed float

commit a9eff1c13405e19285662cc14349e9a20d20ace8
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Mon Nov 6 15:11:50 2017 -0500
added floats, chars, strings, lists to scanner

commit 73b7f9b30dacea28a1cb86935b1f7b99ba72434d4
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Oct 29 18:04:02 2017 -0400
Added operators to ast

commit bc3cb0f0dae025b2c1b656a2fd8dd657661c0
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Oct 29 17:48:39 2017 -0400
added our types to ast

commit cb4fb1fd683fcf84742b188dd9f2fa6fb5987983
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Oct 29 16:56:50 2017 -0400
revert

commit a335920f4532751773f7dd13ede9c0150eb0b6ea1a
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Oct 29 16:56:22 2017 -0400
Revert to initial commit

commit 8513b09308e175e5ba47d5bfd375d96bf7f6af69
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Oct 29 16:47:59 2017 -0400
test

commit a4317c703a7f26bf03ce165b0ed1ad4f415f7f01
Author: Gabriel Kramer-Garcia <glk2110@columbia.edu>
Date: Sun Oct 29 16:44:56 2017 -0400
replace microc

commit ce4ad439ac21b573be3a25abcac5934129e2d3d3
Author: Brian Tsau <bt2420@columbia.edu>
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 pixelformer 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

```python
1  def int main()
2    {
3      print_string("Hello World!");
4      return 0;
5    }
```

LLVM Code Generated

```python
1  define i32 @main() {
2    entry:
3      %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))
4      ret i32 0
5  }
```

6.1.2 Sample Program 2

Adding two vectors in pixelman

```python
1  def int main()
2    {
3      int [3] a;
4      int [3] b;
5      int i;
6      a = [1,2,3];
7      b = [1,1,1];
8      a = a + b;
9      for (i = 0; i < 3; i = i + 1) {
10         printf(a[i]);
11      }
12      return 0;
13  }
```

LLVM Code Generated

```python
1  define i32 @main() {
2    entry:
3      %a = alloca [3 x i32]
4      %b = alloca [3 x i32]
5      %i = alloca i32
6      store [3 x i32], [i32 1, i32 2, i32 3], [3 x i32]* %a
7      store [3 x i32], [i32 1, i32 1, i32 1], [3 x i32]* %b
8      %b1 = load [3 x i32], [3 x i32]* %b
9      %a2 = load [3 x i32], [3 x i32]* %a
10     %vec_vec_addi_result = call [3 x i32] @vec_vec_addi([3 x i32] %a2, [3 x i32] %b1)
11     store [3 x i32], %vec_vec_addi_result, [3 x i32]* %a
12     store i32 0, i32* %i
13     br label %while
14    while:
15      %entry
16      %i7 = load i32, i32* %i
17      %tmp8 = icmp slt i32 %i7, 3
```
6.1.3 Sample Program 3
Reversing a vector in pixelman

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

LLVM Code Generated

```c
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
```
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

#!/bin/sh

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

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

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the microc compiler. Usually "/.microc.native"
# Try ".build/microc.native" if ocamlbuild was unable to create a symbolic link.
MICROC="./pixelman.native"
MICROC="_build/microc.native"

# Set time limit for all operations
ulimit -t 30

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

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

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

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

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

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

Check() {
    error=0
    basename=`echo $1 | sed 's/\.*$//'

reffeile=`echo $1 | sed 's/.px//'`

basedir=`'echo $1 | sed 's/\([^/\"]*\)\*/\1/\]'`

`echo -n "$basename..."`

`echo 1>&2`

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

geneneratedfiles=""

`generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.s ${basename}.out" &&`

Run "$MICROC" "<" "$1" ">${basename}.ll" &&

Run "$LLC" "$"{basename}.ll" "n" "${basename}.s" "printbig.o"

"makePic.o" "inputPic.o" &&

Run "./"{basename}.exe >""${basename}.out" &&

Compare ${basename}.out ${reffeile}.out ${basename}.diff

# Report the status and clean up the generated files

if [ $error -eq 0 ] ; then
  if [ $keep -eq 0 ] ; then
    rm -f $generatedfiles
  fi

  echo "OK"

  echo "######## SUCCESS" 1>&2

else
  echo "######## FAILED" 1>&2

  globalerror=$error

fi

CheckFail() {
  error=0

  basename=`echo $1 | sed 's/\([^/\"]*\)\*/\1/\]'`

  reffeile=`echo $1 | sed 's/.px//'`

  basedir=`'echo $1 | sed 's/\([^/\"]*\)\*/\1/\]'`

  `echo -n "$basename..."`

  `echo 1>&2`

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

  generatedfiles=""

  `generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&`

  RunFail "$MICROC" "<" "$1" "2" ">${basename}.err" ">" "$globallog &&

  Compare ${basename}.err ${reffeile}.err ${basename}.diff

# Report the status and clean up the generated files

if [ $error -eq 0 ] ; then
  if [ $keep -eq 0 ] ; then
    rm -f $generatedfiles
  fi

  echo "OK"

  echo "######## SUCCESS" 1>&2

else
  echo "######## FAILED" 1>&2

  globalerror=$error

  `echo 1>&2`

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

  generatedfiles=""

  `generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.out" &&`

  Run "$MICROC" "<" "$1" ">${basename}.ll" &&

  Run "$LLC" "$"{basename}.ll" "n" "${basename}.s" "printbig.o"

  "makePic.o" "inputPic.o" &&

  Run "./"{basename}.exe >""${basename}.out" &&

  Compare ${basename}.out ${reffeile}.out ${basename}.diff

# Report the status and clean up the generated files

if [ $error -eq 0 ] ; then
  if [ $keep -eq 0 ] ; then
    rm -f $generatedfiles
  fi

  echo "OK"

  echo "######## SUCCESS" 1>&2

else
  echo "######## FAILED" 1>&2

  globalerror=$error

  `echo 1>&2`

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

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

shift `expr $OPTIND − 1`

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

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

if [ ! −f printbig.o ]
then
  echo "Could not find printbig.o"
  echo "Try \"make printbig.o\""
  exit 1
fi

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

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

exit $globalerror

6.2.2 Output of running test suite

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

$ ./testall.sh
−n test−add1...
OK
−n test−arith1...
OK
−n test−arith2...
OK
−n test−arith3...
OK
−n test−ast−arrays...
OK
−n test−astdatatypes...
OK
−n test−astfloat...
OK
−n test−astop...
OK
−n test−castfloattoint...
OK
−n test−castinttofloat...
OK
−n test−codegen−matrixliteral...
OK
−n test−codegen−vector...
OK
−n test−codegenoperators...
OK
−n test−fib...
OK
−n test−floatopsnospace...
OK
−n test−for1...
OK
−n test−for2...
OK
−n test−func1...
OK
−n test−func2...
OK
−n test−func3...
OK
−n test−func4...
OK
−n test−func5...
OK
−n test−func6...
OK
−n test−func7...
OK
−n test−func8...
OK
−n test−gcd...
OK
−n test−gcd2...
OK
−n test−global1...
OK
−n test−global2...
OK
−n test−global3...
OK
−n test−hello...
OK
−n test−if1...
OK
−n test−if2...
67 -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

50
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 ...
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

1 (* Abstract Syntax Tree and functions for printing it
2   * Anthony Chan
3   * Gabriel Kramer-Garcia
4   * Brian Tsau
5   * Teresa Choe
6   *)
7
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | And | Or | Shiftleft | Shiftright | Bitand | Bitor | Bitxor | Mod
8
type uop = Neg | Not | IntCast | FloatCast
9
type expr =
10   Int_Literal of int
11   | Float_Literal of float
12   | Char_Literal of char
13   | String_Literal of string
14   | Vector_Literal of expr list
15   | Matrix_Literal of expr list list
16   | Bool_Literal of bool
17   | Id of string
18   | Binop of expr * op * expr
19   | Unop of uop * expr
20   | Assign of expr * expr
21   | Call of string * expr list
22   | VecAccess of string * expr
23   | MatAccess of string * expr * expr
24   | MatRow of string * expr
25   | MatCol of string * expr
26   | SizeOf of string
27   (* | ImAccess of string * int*)
28   | Noexpr
29
30 type typ =
31   Int
32   | Bool
33   | Float
34   | Char
35   | String
36   | Void
37   (* | Image of expr * expr *)
38   | Vector of typ * expr
39   | Matrix of typ * expr * expr
40
41 type bind = typ * string
42
type stmt =
43   Block of stmt list
44   | Expr of expr
45   | Return of expr
46   | If of expr * stmt * stmt
47   | For of expr * expr * expr * stmt
48   | While of expr * stmt
49
50
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    -> "\leq"
78      Greater -> "\geq"
79      Geq    -> "\geq"
80      Or     -> "||"
81      Mod    -> "\%"
82      Shiftleft  -> "\ll"
83      Shiftright -> "\gg"
84      Bitand  -> "\&"
85      Bitor   -> "\|"
86      Bitxor  -> "\^"
87
88  let string_of_uop = function
89      Neg    -> "-"
90      Not    -> "!
91      IntCast -> "(Int)"
92      FloatCast -> "(Float)"
93
94  let rec string_of_vector el =
95      "]", List.map (fun e -> string_of_expr e) el, "[
96
97  and(* rec *) string_of_matrix el = "[" \String.concat ", " (List.map (fun v -> string_of_vector v) el) "|
98
99  and(* rec *) string_of_expr = function
100      Int_Literal(i)  -> string_of_int i
101      Float_Literal(f) -> string_of_float f
102      Char_Literal(c) -> Char.escaped c
103      String_Literal(s) -> s
104      Bool_Literal(b) -> if b then "true" else "false"
105      Vector_Literal(el) -> string_of_vector el
106      Matrix_Literal(el) -> string_of_matrix el
107      Id(s)            -> s
108      | Binop(e1, o, e2)  -> string_of_expr e1 " " o " " string_of_expr e2
109      | Unop(o, e)         -> string_of_uop o " " string_of_expr e
110      | Assign(e1, e2)     -> string_of_expr e1 " = " string_of_expr e2
111  (* | Assign(v, e2)     -> v " = " string_of_expr e2 *)
let rec string_of_stmt = function
  | Block(stmts) -> "\{\n    " (List.map string_of_stmt stmts) " \n  \n  ")\n
  | Return(expr) -> "return " (List.map string_of_expr expr) "\n
  | If(e, s, Block([])) -> "if " (List.map string_of_expr e) "\n
  | If(e, s1, s2) -> "if " (List.map string_of_expr e) " else \n
  | For(e1, e2, e3, s) -> "for " (List.map string_of_expr e1) "; " (List.map string_of_expr e2) "; " (List.map string_of_expr e3) "\n
  | While(e, s) -> "while " (List.map string_of_expr e) "\n
  | Break -> "break;"\n
  | Continue -> "continue;"\n
let rec string_of_typ = function
  | Int -> "int"\n
  | Bool -> "bool"\n
  | Char -> "char"\n
  | Float -> "float"\n
  | String -> "string"\n
  | Void -> "void"\n
  | Vector(t, e) -> string_of_typ t "[" (List.map string_of_expr e) "]"\n
  | Matrix(t, e1, e2) -> string_of_typ t "[" (List.map string_of_expr e1) " \n
  | "\n
  | Image(h, w) -> "Image[" (List.map string_of_expr h) " , " (List.map string_of_expr w) "]"\n
let string_of_vdecl (t, id) = string_of_typ t " " id " ;\n
let string_of_fdecl fdecl =
  "def " (List.map string_of_typ fdecl.typ) " "\n
  fdecl.fname " (" (List.map string_of_expr fdecl.formals) "\n
  "\n
let string_of_program (vars, funcs) =
  String.concat " (List.map string_of_vdecl vars) " \n
  String.concat " (List.map string_of_fdecl funcs)\n
let rec string_of_stmt = function
  | SizeOf(s) -> "sizeof(" s "\n
  | Call(f, e1) -> "f \n
  | MatAccess(v, e1, e2) -> v "\n
  | MatRow(v, e) -> v "\n
  | MatCol(v, e) -> v "\n
  | ImAccess(v, c) -> v ".\n
  | Noexpr -> ""
### 8.2 codegen.ml

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

* http://llvm.org/docs/tutorial/index.html
* http://llvm.moe/
* http://llvm.moe/ocaml/
* Teresa Choe
* Brian Tsau
* Anthony Chan
* Gabriel Kramer–Garcia
*)

module L = LLVM
module A = Ast
module S = Sast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "Pixelman"
  and i32_t = L.i32_type context
  and i8_t = L.i8_type context
  and il_t = L.i1_type context
  and f_t = L.double_type context
  and array_t = L.array_type
  and void_t = L.void_type context in

  let int_lit_to_int = function
    A.Int.Literal(i) -> i | _ -> raise(Failure("Can only make vector/
      matrix of dimension int literal"))
  in

  let ltype_of_typ = function
    A.Int -> i32_t
    | A.Float -> f_t
    | A.Bool -> il_t
    | A.Char -> i8_t
    | A.String -> i32_t
    | A.Void -> void_t
    | A.Vector(typ, size) -> (match typ with
      A.Int -> array_t i32_t (int_lit_to_int size)
    | A.Float -> array_t f_t (int_lit_to_int size)
      | _ -> raise(Failure("Can only make vector of type int/float")))
    | A.Matrix(t, s1, s2) -> (match t with
      A.Int -> array_t (array_t i32_t (int_lit_to_int s2)) (int_lit_to_int s1)
    | A.Float -> array_t (array_t f_t (int_lit_to_int s2)) (int_lit_to_int s1)
      | _ -> raise(Failure("Cannot only make
        vector of type int/float")))
    (*| A.Image(h, w) -> let mat_t = ltype_of_typ (A.Matrix(A.Int, h, w))
      in array_t mat_t 3 (* make an array of 3 h x w
      matrices *))*)

  (* Declare each global variable; remember its value in a map *)
  let global_vars =
    let global_vars (t, n) =
      let init = if ltype_of_typ t != f_t then
        L.const_int (ltype_of_typ t) 0
```
else

L.const_float (ltype_of_typ t) 0.0
in StringMap.add n (L.define_global n init the_module) m in
List.fold_left global_var StringMap.empty globals in

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

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

let makePic_t = L.function_type i32_t [] i32_t; i32_t; i32_t; i32_t; i32_t in
let makePic_func = L.declare_function "makePic" makePic_t the_module in

let inputPic_t = L.function_type i32_t [] i32_t in
let inputPic_func = L.declare_function "inputPic" inputPic_t the_module in

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

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

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

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

in

let formals = List.fold_left2 add_formal StringMap.empty fdecl.S.sformals  
  (Array.to_list (L.params the_function))

in

List.fold_left add_local_formals fdecl.S.slocals

in

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

in

let rec get_vector_acc_addr s e1 builder = L.build_gep (lookup s)  
  [[L.const_int i32_t 0]; (expr builder e1)] s builder

and get_matrix_acc_addr s e1 e2 builder = L.build_gep (lookup s)  
  [[L.const_int i32_t 0]; expr builder e1; expr builder e2]] s builder

(* Construct code for an expression; return its value *)
and expr builder = function
  S.SInt_Literal i -> L.const_int i32_t i
  | S.SFloat_Literal f1 -> L.const_float f_t f1
  | S.SChar_Literal c -> L.const_int i8_t (Char.code c)
  | S.SString_Literal s -> L.build_global_stringptr s "s" builder
  | S.SBool_Literal b -> L.const_int i1_t (if b then 1 else 0)
  | S.SVector_Literal (l, t) -> L.const_array (ltype_of_typ t) (Array.of_list (List.map (expr_builder) l))
  | S.SMatrix_Literal (ell, t) -> (match t with
      A.Matrix(A.Float, ..) ->
        let order = List.map List.rev ell in
        let f_lists = List.map (List.map (expr_builder)) order in
        let array_list = List.map (List.map f_lists) in
        let f_list_list = List.map (L.const_array f_t) array_list
        in
        let array_of_arrays = Array.of_list f_list_list in
        L.const_array(array_t f_t (List.length (List.hd ell)))
        array_of_arrays
      | A.Matrix(A.Int, ..) ->
        let order = List.map List.rev ell in
        let i_lists = List.map (List.map (expr_builder)) order in
        let array_list = List.map (List.map i_lists) in
        let i_list_array = List.map (L.const_array i32_t) array_list
        in
        let array_of_arrays = Array.of_list i_list_array in
        L.const_array(array_t i32_t (List.length (List.hd ell)))
        array_of_arrays
      | .. -> raise (Failure(A.string_of_typ t))
    )
  | S.SNoexpr -> L.const_int i32_t 0
  | S.SId (s, ..) -> L.build_load (lookup s) s builder
  | S.SSizeOf (vm, ..) -> L.const_int i32_t (L.array_length (L.element_type (L.type_of (lookup vm))))
  | S.SVecAccess (s, e, ..) -> L.build_load (get_vector_acc_addr s e builder) s builder
  | S.SMatAccess (s, e1, e2, ..) -> L.build_load (get_matrix_acc_addr s e1 e2 builder) s builder
  | S.SBinop (e1, op, e2, ..) -> (* too late to implement using sexpr types to make things easier *)
let e1' = expr builder e1
and e2' = expr builder e2 in
(match op with
(* A.Add    -> L.build_add *)
A.Add    -> (let el_type_string = L.string_of_lltype (L.type_of el')
           in
           (match el_type_string with
            "double"  -> L.build_fadd
            | "i32"    -> L.build_add
            | _        -> raise(Failure("Illegal type operation"))) )
| A.Sub    -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fsub
               | "i32"    -> L.build_sub
               | _        -> raise(Failure("Illegal type operation"))) )
| A.Mod    -> L.build_urem
| A.Mult   -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fmul
               | "i32"    -> L.build_mul
               | _        -> raise(Failure("Illegal type operation"))) )
| A.Div    -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fdiv
               | "i32"    -> L.build_sdiv
               | _        -> raise(Failure("Illegal type operation"))) )
| A.And    -> L.build_and
| A.Or     -> L.build_or
| A.Bitxor -> L.build_xor
| A.Bitand -> L.build_and
| A.Bitor  -> L.build_or
| A.Shiftright -> L.build_lshr
| A.Shiftleft -> L.build_shl
| A.Equal  -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fcmp L.Fcmp.Oeq
               | "i32"    -> L.build_icmp L.Icmp.Eq
               | _        -> raise(Failure("Illegal type operation"))) )
| A.Neq    -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fcmp L.Fcmp.One
               | "i32"    -> L.build_icmp L.Icmp.Ne
               | _        -> raise(Failure("Illegal type operation"))) )
| A.Less   -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fcmp L.Fcmp.Olt
               | "i32"    -> L.build_icmp L.Icmp.Slt
               | _        -> raise(Failure("Illegal type operation"))) )
| A.Leq    -> (let el_type_string = L.string_of_lltype (L.type_of el')
              in
              (match el_type_string with
               "double"  -> L.build_fcmp L.Fcmp.Ole
               | "i32"    -> L.build_icmp L.Icmp.Sle
               | _        -> raise(Failure("Illegal type operation"))) )
(match e1.type_string with
  "double" → L.build_fcmp L.Fcmp.Ole
  | "i32" → L.build_icmp L.Icmp.Sle
  | _ → raise(Failure("Illegal type operation "))
)

| A.Greater → (let e1.type_string = L.string_of.lltype (L.type_of e1) in
  (match e1.type_string with
   "double" → L.build_fcmp L.Fcmp.Ogt
   | "i32" → L.build_icmp L.Icmp.Sgt
   | _ → raise(Failure("Illegal type operation "))
  ))

| A.Geq → (let e1.type_string = L.string_of.lltype (L.type_of e1) in
  (match e1.type_string with
   "double" → L.build_fcmp L.Fcmp.Oge
   | "i32" → L.build_icmp L.Icmp.Sge
   | _ → raise(Failure("Illegal type operation "))
  ))

) e1 e2 "tmp" builder

| S.SUnop(op, e, t) → let e' = expr builder e in
  (match op with
   A.Neg → (if t == A.Float then L.build_fneg else
     L.build_neg) e' "tmp" builder
   | A.Not → L.build_not e' "tmp" builder
   | A.IntCast → L.build_fptosi e' i32.t "float_to_int"
     builder
   | A.FloatCast → L.build_sitofp e' f.t "int_to_float"
     builder
  )

| S.SAssign (v, e, _) → let lsb = (match v with
  S.SId(n, _) → lookup n
  | S.SVecAccess(s, e, _) →
    get_vector_acc_addr s e builder
  | S.SMatAccess(s, e1, e2, _) →
    get_matrix_acc_addr s e1 e2 builder
  | _ → raise(Failure("Illegal
    assignment lvalue")))

  in
  let rsb = expr builder e in
  ignore (L.build_store rsb lsb builder); rsb

| SSCALL ("print_int", [e], _) →
  L.build_call printf_func [] int.format_str ; (expr
  builder e) []
  "printf" builder

| SSCALL ("print_string", [e], _) →
  L.build_call printf_func [] string.format_str ; (expr
  builder e) []
  "print_string" builder

| SSCALL ("print_newline", [], _) →
  L.build_call printf_func [] nl.format_str []
  "print_newline" builder

| SSCALL ("print_float", [e], _) →
  L.build_call printf_func [] float.format_str ; (expr
  builder e) []
  "print_float" builder

| SSCALL ("printbig", [e], _) →
  L.build_call printfbig_func [] (expr builder e) []
  "printfbig" builder

| SSCALL ("makePic", [e:e1:e2:e3:e4], _) →
  L.build_call makePic_func [] (expr builder e);(expr builder e1)
  ;(expr builder e2);(expr builder e3);(expr builder e4) []
  "makePic" builder
| SSCALL ("inputPic", [e], _) ->  
| L.build_call inputPic_func [[] (expr builder e) []] "inputPic" builder 
| SSCALL (f, act, _) ->  
| let (fdef, fdecl) = StringMap.find f function_decls in 
| let actuals = List.rev (List.map (expr builder) (List.rev act)) in 
| let result = (match fdecl.S.styp with A.Void -> "" 
| | _ -> f `""_result") in 
| L.build_call fdef (Array.of_list actuals) result builder 

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

(* Build the code for the given statement; return the builder for the statement’s successor *) 
let rec stmt builder = function 
| S.SBlock sl -> List.fold_left stmt builder sl 
| S.SExpr e -> ignore (expr builder e); builder 
| S.SReturn e -> ignore (match fdecl.S.styp with 
| A.Void -> L.build_ret_void builder 
| | _ -> L.build_ret (expr builder e) builder); builder 
| S.SIf (predicate, then_stmt, else_stmt) -> 
| let bool_val = expr builder predicate in 
| let merge_bb = L.append_block context "merge" the_function in 
| add_terminal (stmt (L.builder_at_end context then_bb) then_stmt) 
| (L.builder merge_bb); 
| let else_bb = L.append_block context "else" the_function in 
| add_terminal (stmt (L.builder_at_end context else_bb) else_stmt) 
| (L.builder merge_bb); 
| ignore (L.build_cond_br bool_val then_bb else_bb builder); 
| L.builder_at_end context merge_bb 

| S.SWhile (predicate, body) ->  
| let pred_bb = L.append_block context "while" the_function in 
| ignore (L.build_br pred_bb builder); 
| let body_bb = L.append_block context "while_body" the_function in 
| add_terminal (stmt (L.builder_at_end context body_bb) body) 
| (L.builder_br pred_bb); 
| let pred_builder = L.builder_at_end context pred_bb in 
| let bool_val = expr pred_builder predicate in 
| let merge_bb = L.append_block context "merge" the_function in 
| ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder) 
| (L.builder merge_bb); 

| S.SFor (e1, e2, e3, body) -> stmt builder 
| (S.SBlock [S.SExpr e1 ; S.SWhile (e2, S.SBlock [body ; S.

61
let builder = stmt builder (S.SBlock fdecl.S.sbody) in

(* Add a return if the last block falls off the end *)
add_terminal builder (match fdecl.S.styp with
  A.Void -> L.build_ret_void
| t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in
List.iter build_function_body functions;
the_module
8.3 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 dimensions in pixels
10     static int image[300][300][4]; // first [ ] number here is total pixels
11         // of each color in my image, 3 is for //RGB values
12     FILE *streamIn;
13     // // opening 24 bit image
14     streamIn = fopen("edwards.bmp", "r"); // a bigger star in black and a
15         // // star in blue (refer
16             // figure attached)
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
21     }
22     // // initiating with new "i" different from above
23     int k;
24     for(k=0;k<pix_y;k++) {
25         for(j=0;j<pix_x;j++) {
26             image[k][j][3] = fgetc(streamIn);
27             image[k][j][2] = fgetc(streamIn); // use BMP 24bit with no alpha
28                 // channel
29             image[k][j][1] = fgetc(streamIn); // BMP uses BGR but we want RGB
30                 // grab //byte−by−byte
31         image[k][j][0] = fgetc(streamIn);
32     }
33     }
34     if(effect==1){
35         int temp;
36         for(k=0;k<pix_y;k++) {
37             for(j=0;j<pix_x;j++) {
38                 temp = (image[k][j][2] + image[k][j][1] + image[k][j][0]) / 3;
39             }
40         }
41     }
42     if(effect==2){
43         float tempR);
44     float tempG;
45     float tempB;
46     for(k=0;k<pix_y;k++) {
47         for(j=0;j<pix_x;j++) {
48             tempR = image[k][j][2] *.189 + image[k][j][1] *.769 + image[k][j]
49                 ||[0] *. 393;
50             tempG = image[k][j][2] *.168 + image[k][j][1] *.686 + image[k][j]
51                 ||[0] *. 349;
52             tempB = image[k][j][2] *.131 + image[k][j][1] *.534 + image[k][j]
53                 ||[0] *. 272;
54     }
image[k][j][2] = new_min((int) tempB, 255);
image[k][j][1] = new_min((int) tempG, 255);
image[k][j][0] = new_min((int) tempR, 255);
}

FILE *fp = fopen("outimage.ppm", "wb"); /* b = binary mode */
(void) fprintf(fp, "P6\n%d %d\n%255\n", pix_x, pix_y);
for(k=0;k<pix_y;k++) {
  for(j=0;j<pix_x;j++) {
    static unsigned char color[3];
    color[0] = image[k][j][0]; /* red */
    color[1] = image[k][j][1]; /* green */
    color[2] = image[k][j][2]; /* blue */
    (void) fwrite(color, 1, 3, fp);
  }
}
(void) fclose(fp);
8.4 Makefile

1 # Make sure ocamlbuild can find opam-managed packages: first run
2 # eval 'opam config env'
3 # Easiest way to build: using ocamlbuild, which in turn uses ocamlfind
4 all : clean pixelman.native printbig.o makePic.o inputPic.o
5
6 pixelman.native :
7     ocamlbuild -use-ocamlfind -pkg llvm,llvm.analysis -cflags -w,+a
8         -4 \    
9         pixelman.native
10
11 # "make clean" removes all generated files
12 .PHONY : clean
13 clean :
14     ocamlbuild -clean
15     rm -rf testall.log *.diff pixelman scanner.ml parser.ml parser.mli
16     rm -rf printbig makePic inputPic
18
19 # More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM
20 OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx pixelman.cmx
21
22 pixelman : $(OBJS)
23     ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis
24         $(OBJS) -o pixelman
25
26 scanner.ml : scanner.mll
27     ocamllex scanner.mll
28
29 parser.ml parser.mli : parser.mly
30     ocamlyacc parser.mly
31
32 %.cmo : %.ml
33     ocamlc -c $<
34
35 %.cmi : %.mli
36     ocamlc -c $<
37
38 %.cmx : %.ml
39     ocamlfind ocamlopt -c -package llvm $<
40
41 # Testing the "printbig" example
42 printbig : printbig.c
43     cc -o printbig -DBUILD_TEST printbig.c
44
45 makePic : makePic.c
46     cc -o makePic -DBUILD_TEST makePic.c
47
48 inputPic : inputPic.c
49     cc -o inputPic -DBUILD_TEST inputPic.c
50
51 ### Generated by "ocamldep *.ml *.mli" after building scanner.ml and
52     parser.ml
53 ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
pixelman.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
pixelman.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

# Building the tarball

TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3
  func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3
  hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2
  while1 while2 printbig
FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2
  for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8
  func9 global1 global2 if1 if2 if3 nomain return1 return2 while1
  while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

TARFILES = ast.ml codegen.ml Makefile pixelman.ml parser.mly README
  scanner.ml $(TESTFILES:%=tests/%) printbig.c makePic.c
  inputPic.c arcade–font.pbm $(font2c
pixelman.tar.gz : $(TARFILES)
  cd .. && tar czf pixelman/pixelman.tar.gz $(TARFILES:%=pixelman/%)
8.5 makePic.c

/*
Part of our C library
Created by Gabriel Kramer-Garcia
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void makePic(int width, int height, int red, int green, int blue) {
    const int dimx = width, dimy = height;
    int i, j;
    FILE *fp = fopen("first.ppm", "wb"); /* b - binary mode */
    (void) fprintf(fp, "P6\n%d %dn255\n", dimx, dimy);
    for (j = 0; j < dimy; ++j) {
        for (i = 0; i < dimx; ++i) {
            static unsigned char color[3];
            color[0] = red; /* red */
            color[1] = green; /* green */
            color[2] = blue; /* blue */
            (void) fwrite(color, 1, 3, fp);
        }
    }
    (void) fclose(fp);
}
8.6 parser.mly

/* Ocamlyacc parser for Pixelman
 * Teresa Choe
 * Anthony Chan
 * Brian Tsau
 * Gabriel Kramer-Garcia
 */

{%
open Ast
%
%
%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA COLON DOT
%token LMATBRACK RMATBRACK
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT INTCAST FLOATCAST
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT FLOAT BOOL VOID DEF STRING CHAR
   SIZEOF /*IMAGE*/
%
%token NOVECLBRACKET
%token BREAK CONTINUE
%token LSHIFT RSHIFT BITAND BITXOR BITOR MOD
%token <int> INT_LITERAL
%token <string> ID
%token <char> CHAR_LITERAL
%token <float> FLOAT_LITERAL
%token <string> STRING_LITERAL
%token EOF
%
%nonassoc NOELSE
%nonassoc ELSE
%nonassoc NOVECLBRACKET
%nonassoc LMATBRACK
%nonassoc LBRACKET
%nonassoc DOT
%right ASSIGN
%left OR
%left AND
%left BITOR
%left BITXOR
%left BITAND
%left EQ NEQ
%left LT GT LEQ GEQ
%left LSHIFT RSHIFT
%right INTCAST FLOATCAST
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT NEG
%
%start program
%type <Ast.program> program
%
%%
%
program:
 decls EOF { $1 }

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

DEF typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE

{ typ = $2;
  fname = $3;
  formals = $5;
  locals = List.rev $8;
  body = List.rev $9 }

formals_opt:

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

formal_list:

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

typ:

    INT { Int } | BOOL { Bool } | FLOAT { Float } | CHAR { Char } | STRING { String } | VOID { Void }
    /* im_t { $1 } */ | vec_t { $1 } | mat_t { $1 }

vdecl_list:

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

typ

    vdecl:

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

vec_t:

typ LBRACKET expr RBRACKET %prec NOVECLBRACKET { Vector($1, $3) }

mat_t:

ty LBRACKET expr RBRACKET LBRACKET expr RBRACKET { Matrix($1, $3, $6) }

/* im_t: */

IMAGE LBRACKET expr COMMA expr RBRACKET %prec NOVECLBRACKET { Image($3, $5) } */

stmt_list:

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

stmt:

    expr SEMI { Expr $1 } | RETURN SEMI { Return Noexpr } | RETURN expr SEMI { Return $2 } | LBRACE stmt_list RBRACE { Block(List.rev $2) } | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) } | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) } | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt { For($3, $5, $7, $9) } | WHILE LPAREN expr RPAREN stmt { While($3, $5) } /* | BREAK SEMI { Break } */
expr_opt:  
/∗ nothing ∗/  { Noexpr }  
|  expr  
|  { $1 }  

expr:  
literals  { $1 }  
|  ID  
|  expr PLUS expr  { Binop($1, Add, $3) }  
|  expr MINUS expr  { Binop($1, Sub, $3) }  
|  expr TIMES expr  { Binop($1, Mult, $3) }  
|  expr DIVIDE expr  { Binop($1, Div, $3) }  
|  expr MOD expr  { Binop($1, Mod, $3) }  
|  expr EQ expr  { Binop($1, Equal, $3) }  
|  expr LEQ expr  { Binop($1, Leq, $3) }  
|  expr GT expr  { Binop($1, Greater, $3) }  
|  expr GEQ expr  { Binop($1, Geq, $3) }  
|  expr AND expr  { Binop($1, And, $3) }  
|  expr OR expr  { Binop($1, Or, $3) }  
|  expr LSHIFT expr  { Binop($1, Shiftleft, $3) }  
|  expr RSHIFT expr  { Binop($1, Shiftright, $3) }  
|  expr BITAND expr  { Binop($1, Bitand, $3) }  
|  expr BITOR expr  { Binop($1, Bitor, $3) }  
|  expr BITXOR expr  { Binop($1, Bitxor, $3) }  
|  expr ASSIGN expr  { Assign($1, $3) }  
|  SIZEOF LPAREN ID RPAREN  { SizeOf($3) }  
|  ID LPAREN actuals opt RPAREN  { Call($1, $3) }  
|  INTCAST expr  { IntCast($2) }  
|  FLOATCAST expr  { FloatCast($2) }  
|  expr ASSIGN expr  { Assign($1, $3) }  

primitive_literals:  
|  INT_LITERAL  { Int_Literal($1) }  
|  STRING_LITERAL  { String_Literal($1) }  
|  FLOAT_LITERAL  { Float_Literal($1) }  
|  CHAR_LITERAL  { Char_Literal($1) }  
|  TRUE  { Bool_Literal(true) }  
|  FALSE  { Bool_Literal(false) }  

literals:  
|  primitive_literals  { $1 }  
|  LBRACKET array_literal RBRACKET  { Vector_Literal(List.rev $2) }  
|  LMATBRACK matrix_literal RMATBRACK  { Matrix_Literal(List.rev $2) }  

matrix_literal:  
|  LBRACKET array_lITERAL RBRACKET  { [$2] }  
|  matrix_literal BITAND LBRACKET array_literal RBRACKET  { $4 :: $1 }  

array_literal:  
|  primitive_literals  { $1 }  
|  array_literal COMMA primitive_literals  { $3 :: $1 }  


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

actuals_list:
   expr { [$1] }
   | actuals_list COMMA expr { $3 :: $1 }
8.7 pixelman.ml

(*check the resulting AST, generate LLVM IR, and dump the module *)
(*Linking to our standard library
Gabriel Kramer–Garcia
*)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);   (* Print the AST only *)
                        ("-l", LLVM_IR); (* Generate LLVM, don’t
                          check *)
                        ("-c", Compile) ] (* Generate, check LLVM
                          IR *)
  else Compile in

let file_to_string file =
  let array_string = ref [] in
  let ic = file in
  try
    while true do
      array_string := List.append !array_string [input_line ic]
        done;
    String.concat "\n" !array_string
  with End_of_file -> close_in ic; String.concat "\n" !
    array_string

in

let in_file = open_in "stdlib.px" in
let string_in = file_to_string in_file in
let other_file = file_to_string stdin in
let str = String.concat "\n" [other_file; string_in] in

let lexbuf = Lexing.from_string str in
let ast = Parser.program Scanner.token lexbuf in
let sast = Semant.check ast in
match action with
  Ast -> print_string (Ast.string_of_program ast)
| LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate
                          sast))
| Compile -> let m = Codegen.translate sast in
            Llvm_analysis.assert_valid_module m;
          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
3 LLVM
4 */
5
6 #include <stdio.h>
7
8 /*
9 * Font information: one byte per row, 8 rows per character
10 * In order, space, 0−9, A−Z
11 */
12
13 static const char font[] = {
14 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
15 0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
16 0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
17 0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
18 0x20, 0x61, 0x49, 0x4d, 0x7f, 0x7f, 0x31, 0x00,
19 0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
20 0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
21 0x3c, 0x7e, 0x4b, 0x41, 0x41, 0x63, 0x22, 0x00,
22 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
23 0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
24 0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
25 0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
26 0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
27 0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
28 0x7f, 0x7f, 0x08, 0x08, 0x08, 0x08, 0x7f, 0x7f,
29 0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
30 0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
31 0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
32 0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x00,
33 0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
34 0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
35 0x3e, 0x7f, 0x41, 0x41, 0x7f, 0x3e, 0x00, 0x00,
36 0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
37 0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
38 0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
39 0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
40 0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
41 0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
42 0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
43 0x1f, 0x38, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
44 0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
45 0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
46 0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
47 
48 };  
49 
50 void printbig(int c)  
51 {  
52   int index = 0;  
53   int col, data;  
54   if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;  
55   else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;  
56   do {  
57     data = font[index++];  
58     for (col = 0 ; col < 8 ; data <<= 1, col++) {  
59       char d = data & 0x80 ? 'X' : ' ';  
60     }
61   }
62   else {
63     data = 0;
64     for (char d = 0; d < 8; data <<= 1) {
65       if (data & 0x80) d = 'X';
66     }
67   }  
68   for (char *s = font; s <= font + index; s++)
69     putchar (*s);
70   putchar('
');
putchar(d); putchar(d);
}
putchar(\n);
} while (index & 0x7);
}

#define BUILD_TEST
int main()
{
char s[] = "HELLO WORLD09AZ";
char *c;
for (c = s; *c; c++) printbig(*c);
}
#endif
8.9 sast.ml

(* pixelman's Semantically Checked Abstract Syntax Tree and functions for printing it
   * Gabriel Kramer-Garcia
   * Anthony Chan
   * Teresa Choe
   * Brian Tsau
   *
)

open Ast

(* sexpressions, ssome with sdatatype sas sadditional sinformation *)

type sexpr =
    SInt_Literal of int
| SFloat_Literal of float
| SChar_Literal of char
| SString_Literal of string
| SBool_Literal of bool
| SVector_Literal of sexpr list * typ
| SMatrix_Literal of sexpr list list * typ
| SId of string * typ
| SBinop of sexpr * op * sexpr * typ
| SUnop of uop * sexpr * typ
| SAssign of sexpr * sexpr * typ
| SVecAccess of string * sexpr * typ
| SMatAccess of string * sexpr * sexpr * typ
| (SImAccess of string * int * typ)
| SCall of string * sexpr list * typ
| SSizeOf of string * typ
| SNoexpr

(* statements *)

type sstmt =
    SBlock of sstmt list
| SExpr of sexpr
| SReturn of sexpr
| SIf of sexpr * sstmt * sstmt
| SFor of sexpr * sexpr * sexpr * sstmt
| SWhile of sexpr * sstmt
| (SBreak
| SContinue *)

(* sfunction sdeclarations *)

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

(* sprogram *)

(type sprogram = bind list * func_decl list)
8.10 scanner.mll

(* Ocamlex scanner for pixelman
 * Brian Tsau
 * Teresa Choe
 * Gabriel Kramer-Garcia
 * Anthony Chan
 *

open Parser

let character = [' ' '-!' '#-'][ ':'] ['-'] | ('\\' ['"' ' ''] 'n' 'r' 't'])

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

let whitespace = [' ' '
' ' ' ' ' ']

let float_lit = (digit+) ['.'] digit+

let int_lit = digit+

rule token = parse

whitespace { token lexbuf } (* Whitespace *)
| "::") { sl_comment lexbuf }
| "::" { comment lexbuf }

| '(' { LPAREN }
| ')' { RPAREN }
| '{' { LBRACE }
| '}' { RBRACE }
| ';' { SEMI }
| ',' { COMMA }
| "[" { LBRACKET }
| "]" { RBRACKET }
| '"" { LMATBRACK }
| '"" { RMATBRACK }
| '*' { COLON } *
| ":" { DOT }

| '=' { ASSIGN }

| "$int" { INTCAST }
| "$float" { FLOATCAST }

| '+*' { PLUS }
| '-*' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }
| '%' { MOD }

| "==" { EQ }
| "!=' { NEQ }
| '<' { LT }
| '<=' { LEQ }
| '>' { GT }
| '>=' { GEQ }

| "&&" { AND }
| "||" { OR }

(* Binary Boolean Operators *)
Unary Boolean Operators

- "!" (NOT)

Binary Bitwise Operators

- "|" (BITOR)
- "<" (LSHIFT)
- ">" (RSHIFT)
- "&" (BITAND)
- "^" (BITXOR)

Branching Control

- "if" (IF)
- "else" (ELSE)
- "for" (FOR)
- "while" (WHILE)

- "continue" (CONTINUE)
- "break" (BREAK)
- "return" (RETURN)

Function Definition

- "def" (DEF)

Data and Return Types

- "char" (CHAR)
- "int" (INT)
- "float" (FLOAT)
- "bool" (BOOL)
- "string" (STRING)
- "void" (VOID)
- "true" (TRUE)
- "false" (FALSE)

- "sizeof" (SIZEOF)

Literals

- "intLit" as lxm {INT_LITERAL(int_of_string lxm)}
- "charLit" as lxm {ID(lxm)}
- "strLit" as lxm {STRING_LITERAL(lxm)}
- "floatLit" as lxm {FLOAT_LITERAL(float_of_string lxm)}

Comment

- EOF

Tokenization

- ":;" {token lexbuf}
- ":" {token lexbuf}
- "\n" {token lexbuf}
- ":" {sl_comment lexbuf}
8.11 semant.ml

open Ast
open Sast

module StringMap = Map.Make(String)

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

let check (globals, functions) =

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

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

(** Checking Global Variables ***)
List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;
report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

(** Checking Functions ***)
let protected_functions = ["print_int"; "error"; "scan"; "size"; "load"; "write";
"display"); "resize"); "transform"); "print_float"); "print_string"] in

let rec check_protected = function
  [] -> ()
| h :: t -> if List.mem h (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function " ^ h ^ " may not be defined"))
  else ignore (check_protected t)
in check_protected protected_functions;

report_duplicate (fun n -> "duplicate function " ^ n) (List.map (fun fd -> fd.fname) functions);

(* Function declaration for a named function *)
let built_in_decls = StringMap.add "print_int"
  { typ = Void; fname = "print_int"; formals = [(Int, "x")];
locals = []; body = [] (StringMap.add "printb")
{ typ = Void; fname = "printb"; formals = [(Bool, "x")];
  locals = []; body = [] (StringMap.add "print_newline")
{ typ = Void; fname = "print_newline"; formals = [];
  locals = []; body = [] (StringMap.add "printbig")
{ typ = Void; fname = "printbig"; formals = [(Int, "x")];
  locals = []; body = [] (StringMap.add "inputPic")
{ typ = Void; fname = "inputPic"; formals = [(Int, "x")];
  locals = []; body = [] (StringMap.add "makePic")
{ typ = Void; fname = "makePic"; formals = [(Int, "x"); (Int, "x"); (Int, "x"); (Int, "x"); (Int, "x")];
  locals = []; body = [] (StringMap.add "print_string")
{ typ = Void; fname = "print_string"; formals = [(String, "x")];
  locals = []; body = [] (StringMap.add "print_float")
{ typ = Void; fname = "print_float"; formals = [(Float, "x")];
  locals = []; body = [] })})

in

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

in

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

in

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

let fdecl_to sfdecl func =
  List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^ " in " ^ func.fname)) func.formals;
report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
  (List.map snd func.formals);
report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
  (List.map snd func.locals);

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

let type_of_identifier s =
  try StringMap.find s symbols
  with Not_found -> raise (Failure ("undeclared identifier " ^ s))
in
let access_type = function
  Vector(t, _) -> t
  | Matrix(t, _, _) -> t
  | _ -> raise (Failure ("illegal matrix/vector access"))
in

let is_vec_matrix t = match t with
  Vector(_, _) -> ()
  | Matrix(_, _, _) -> ()
  | _ -> raise (Failure ("cannot get size of non-vector/non-matrix type"))
let get_sexp_type se = match se with
  | SInt_Literal(_) -> Int
  | SFloat_Literal(_) -> Float
  | SChar_Literal(_) -> Char
  | SString_Literal(_) -> String
  | SBool_Literal(_) -> Bool
  | SVector_Literal(_, t) -> t
  | SMatrix_Literal(_, t) -> t
  | SId(_, t) -> t
  | SBinop(_, _, _, t) -> t
  | SUnop(_, _, t) -> t
  | SAssign(_, _, t) -> t
  | SVecAccess(_, _, t) -> t
  | SMatAccess(_, _, _, t) -> t
  | SImAccess(_, _, t) -> t
  | SCCall(_, _, t) -> t
  | SSizeOf(_, t) -> t
  | SNoexpr -> Void
in

let get_binop_boolean_sexp se1 se2 op =
  let t1 = get_sexp_type se1 in
  let t2 = get_sexp_type se2 in
  match (t1, t2) with
    | (Bool, Bool) -> SBinop(se1, op, se2, Bool)
    | _ -> raise (Failure ("can only perform boolean operators with Int/Bool types"))

and get_unop_boolean_sexp se op =
  let t = get_sexp_type se in
  match t with
    | Bool -> SUnop(op, se, Bool)
    | _ -> raise (Failure ("can only perform boolean operators with Int/Bool types"))

and get_binop_arithmetic_sexp se1 se2 op =
  let t1 = get_sexp_type se1 in
  let t2 = get_sexp_type se2 in
  match (t1, t2) with
    | (Int, Int) -> SBinop(se1, op, se2, Int)
    | (Int, Float) -> SBinop(SUnop(FloatCast, se1, Float), op, se2, Float)
    | (Float, Int) -> SBinop(se1, op, SUnop(FloatCast, se2, Float), Float)
    | (Float, Float) -> SBinop(se1, op, se2, Float)
    | (Vector(tml, Int_Literal(i)), ta2) -> (match op with
      | Mult -> (match ta2 with
        | Float -> SCCall("scalar_mult_vecf", [se2; se1], Vector(Float, Int_Literal(i)))
        | Int -> if tml == Float
          then SCCall("scalar_mult_vecf", [se2; se1], Vector(Float, Int_Literal(i)))
          else SCCall("scalar_mult_veci", [se2; se1], Vector(Int, Int_Literal(i)))
        | Vector(tm2, _) -> if tm2 == Float || tml == Float
          then SCCall("vec_dot_productf", [se2; se1], Float)
          else SCCall("vec_dot_producti", [se2; se1], Int)
        | Matrix(tm2, _, Int_Literal(j2)) -> if tm2 == Float
          then SCCall("vec_mat_multf", [se1; se2], Vector(Float, Int_Literal(j2)))
          else SCCall("vec_mat_multi", [se1; se2], Vector(Int, Int_Literal(j2)))
    | _ -> raise (Failure "can only perform arithmetic operators with Int/Float types"))
else SCall("vec_mat_multi", [sel; se2], Vector(Int, Int_Literal(j2)))
   | _ -> raise (Failure ("can only perform binary arithmetic operators with Int/Float variables or matrices"))

Sub -> (match ta2 with
   
   Vector(tm2, Int_Literal(i)) -> if tm2 == Float ||
      tml == Float
      then SCall("vec_subb", [sel; se2], Vector(Float, Int_Literal(i)))
   
   else SCall("vec_subb", [sel; se2], Vector(Int, Int_Literal(i)))
   | _ -> raise (Failure ("oh no! can only perform this operation on vector of same length."))

Add -> (match ta2 with
   
   Vector(tm2, Int_Literal(i)) -> if tm2 == Float ||
      tml == Float
      then SCall("vec_addf", [sel; se2], Vector(Float, Int_Literal(i)))
   
      else SCall("vec_addf", [sel; se2], Vector(Int, Int_Literal(i)))
   | _ -> raise (Failure ("oh no! can only perform this operation on vector of same length."))

   | _ -> raise (Failure ("oh no! cannot perform this operation on vector.")))

| (Matrix(tml, Int_Literal(i), Int_Literal(j)), ta2) -> (match op with
   Mult -> (match ta2 with (* matrix x ta2 check *)
      Float -> if i == 3 then
         SCall("scalar_mult_mat3f", [se2; sel], Matrix(Float, Int_Literal(i), Int_Literal(j)))
      
      else SCall("scalar_mult_mat2f", [se2; sel], Matrix(Float, Int_Literal(i), Int_Literal(j))
   | Int ->
      let sc_mat_f_fname = if i == 2 then "scalar_mult_mat2f" else "scalar_mult_mat3f"
      in
      let sc_mat_i_fname = if i == 2 then "scalar_mult_mat2i" else "scalar_mult_mat3i"
      in
      if tml == Int
         then SCall(sc_mat_i_fname, [se2; sel], Matrix(Int, Int_Literal(i), Int_Literal(j)))
      
      else SCall(sc_mat_f_fname, [se2; sel], Matrix(Float, Int_Literal(i), Int_Literal(j)))

| Vector(tm2, _) -> if tm2 == Float ||
   
   tml == Float
   then SCall("mat_vec_multf", [sel; se2], Vector(Float, Int_Literal(i)))
   
   else SCall("mat_vec_multf", [sel; se2], Vector(Int, Int_Literal(i)))

| Matrix(tm2, _, Int_Literal(j2)) -> if tm2 ==
   
   Float || tml == Float
   then SCall("mat_mat_multf", [sel; se2], Matrix(Float, Int_Literal(i), Int_Literal(j2)))
   
   else SCall("mat_mat_multf", [sel; se2], Matrix(Int, Int_Literal(i), Int_Literal(j2)))

   | _ -> raise (Failure ("can only perform binary arithmetic operators with Int/Float variables or matrices"))

| Sub -> (match ta2 with
Matrix(tm2, Int_Literal(i), Int_Literal(j)) -> if tm2
  == Float || tm1 == Float
  then SCall("mat_subf", [se1; se2], Matrix(Float,
  Int_Literal(i), Int_Literal(j)))
  else SCall("mat_subi", [se1; se2], Matrix(Int,
  Int_Literal(i), Int_Literal(j)))
  | _ -> raise (Failure ("oh no! can only perform this
  operation on matrix of same size."))

Add -> (match ta2 with
  Matrix(tm2, Int_Literal(i), Int_Literal(j)) -> if tm2
  == Float || tm1 == Float
  then SCall("mat_addf", [se1; se2], Matrix(Float,
  Int_Literal(i), Int_Literal(j)))
  else SCall("mat_addi", [se1; se2], Matrix(Int,
  Int_Literal(i), Int_Literal(j)))
  | _ -> raise (Failure ("oh no! can only perform this
  operation on matrix of same size."))

  | _ -> raise (Failure ("oh no! cannot perform this
  operation on matrix.")

  | (Int, Vector(Int, Int_Literal(i))) -> SCall("scalar_mult_veci",
  [se1; se2], Vector(Int, Int_Literal(i)))
  | (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)))
  | (Int, Matrix(Int, Int_Literal(i), Int_Literal(j))) ->
  if i == 2 then SCall("scalar_mult_mat2i", [se1; se2], Matrix
  (Int, Int_Literal(i), Int_Literal(j)))
  else SCall("scalar_mult_mat3i", [se1; se2], Matrix(Int,
  Int_Literal(i), Int_Literal(j)))
  | (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)))
  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)))
  | (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)))
  else SCall("scalar_mult_mat3f", [SUnop(FloatCast, se1, Float
  ); se2], Matrix(Float, Int_Literal(i), Int_Literal(j)))
  | (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)))
  else SCall("scalar_mult_mat3f", [se1; se2], Matrix(Float,
  Int_Literal(i), Int_Literal(j)))
  | _ -> raise (Failure ("can only perform binary arithmetic
  operators with Int/Float variables or matrices")

  | _ -> raise (Failure ("can only perform unary arithmetic
  operators with Int/Float variables or matrices")

  and get_unop_arithmetic_sexp se op =
  let t = get_sexp_type se in
  match t with
  | Int -> SUnop(op, se, Int)
  | Float -> SUnop(op, se, Float)
  | Vector(Int,_) -> SCall("negVectori",[se],t)
  | Vector(Float,_) -> SCall("negVectorf",[se],t)
  | Matrix(Int,_) -> SCall("negMatrixi",[se],t)
  | Matrix(Float,_) -> SCall("negMatrixf",[se],t)
  | _ -> raise (Failure ("can only perform unary arithmetic
  operators with Int/Float variables or matrices"))
and get_binop_bitwise_sexpr se1 se2 op =
    let t1 = get_sexpr_type se1 in
    let t2 = get_sexpr_type se2 in
    match (t1, t2) with
        (Int, Int) -> SBinop(se1, op, se2, Int)
    | _  -> raise (Failure ("can only perform bitwise operations on integer types"))

and get_binop_comparison_sexpr se1 se2 op =
    let t1 = get_sexpr_type se1 in
    let t2 = get_sexpr_type se2 in
    match (t1, t2) with
        (Int, Int) -> SBinop(se1, op, se2, Bool)
    | (Float, Float) -> SBinop(se1, op, se2, Bool)
    | _  -> raise (Failure ("can only compare ints/floats with themselves for inequalities"))

and get_unop_cast se expr op =
    let t1 = get_sexpr_type se in
    let t = match t1 with
        Int | Float -> t1
    | Vector(t2, _) -> t2
    | Matrix(t2, _, _) -> t2
    | _  -> raise (Failure ("can only cast int/float expressions to int/float expressions"))
    in
    let op_t = match op with
        IntCast -> Int
    | FloatCast -> Float
    | _  -> raise (Failure ("this is impossible to reach :"))
    in
    if t == op_t then se else
    match t1 with
        Int -> SUnop(op, se, op_t)
    | Float -> SUnop(op, se, op_t)
        Vector(Int, se1) -> SCall("vec_int_to_float", [se], Vector(op_t, se1))
    | Vector(Float, se1) -> SCall("vec_float_to_int", [se], Vector(op_t, se1))
    | Matrix(Int, se1, se2) -> SCall("mat_int_to_float", [se], Matrix(op_t, se1, se2))
    | Matrix(Float, se1, se2) -> SCall("mat_float_to_int", [se], Matrix(op_t, se1, se2))
    | _  -> raise (Failure ("this is impossible to reach :"))

and get_equality_type se1 se2 op =
    let t1 = get_sexpr_type se1 in
    let t2 = get_sexpr_type se2 in
    match (t1, t2) with
        (Int, Int) -> SBinop(se1, op, se2, Bool)
    | (Float, Float) -> SBinop(se1, op, se2, Bool)
    | (Char, Char) -> SBinop(se1, op, se2, Bool)
    | _  -> raise (Failure ("can only compare ints/floats/chars with themselves for equality"))

(*and check_image_accessor ia =
    match ia with
        0  -> ()
    | 1  -> ()
    | 2  -> ()
    | _  -> raise (Failure ("can only access 0, 1, or 2"))*)

(*and check_is_image id =
let idd = type_of_identifier id in
match idd with
  Image(_,_) -> ()
| _ -> raise (Failure("can only access images"))
*)
in
(* Return an sexpr given an expr *)
let rec expr_to_sexpr = function
  Int_Literal(i) -> SInt_Literal(i)
| String_Literal(s) -> SString_Literal(s)
| Float_Literal(f) -> SFloat_Literal(f)
| Bool_Literal(b) -> SBool_Literal(b)
| Char_Literal(c) -> SChar_Literal(c)
| Vector_Literal(e) -> check_vector_types e
| Matrix_Literal(e1) -> check_matrix_types e1
| Id s -> SId(s, type_of_identifier s)
| SizeOf(s) -> is_vec_matrix (type_of_identifier s); SSSizeOf(s, Int)
| VecAccess(v, e) -> check_int_expr e; check_vec_access_type v; SVecAccess(v, expr_to_sexpr e, access_type (type_of_identifier v))
| 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))
| MatRow(s, e) -> check_int_expr e; check_mat_access_type s; get_mat_row_sexpr s e
| MatCol(s, e) -> check_int_expr e; check_mat_access_type s; get_mat_col_sexpr s e
(* ImAccess(idd, color) -> ignore(check_image_accessor color); ignore(check_is_image idd); SImAccess(idd, color, (type_of_identifier idd))*)
| Binop(e1, op, e2) (* as e *) -> get_binop_sexpr e1 e2 op
| Unop(op, e) (* as ex *) -> get_unop_sexpr op e
| Noexpr -> SNoexpr
| Assign(var, e) (* as ex *) -> get_assign_sexpr var e
(* | Call() *) (* let's put the std library functions in here *)
| Call(fname, actuals) as call -> let fd = function_decl fname in
  if List.length actuals != List.length fd.formals then raise (Failure("expecting " ^ string_of_int
  (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
else
  SCall(fname, List.map2 (fun (ft,_) e -> let se =
      let e' = expr_to_sexpr e (* some implicit int -> float conversion done here *)
    in let et2 = get_sexpr_type e'
  in match ft with
    Float -> if get_sexpr_type e' = Int then SUnop(
      FloatCast, e', Float) else e'
    | Vector(Float,_) -> (match et2 with
      Vector(Float,_) -> e'
    | Vector(Int,Int_Literal(_)) -> (expr_to_sexpr (Unop
      (FloatCast, e)))
    | _ -> raise (Failure("can only have vector of int/float")))
  | Matrix(Float,_) -> (match et2 with
    Matrix(Float,_) -> e'
  | Matrix(Int,Int_Literal(_),Int_Literal(_)) -> (expr_to_sexpr (Unop(FloatCast, e)))
  | _ -> raise (Failure("can only have vector of int/float")))
let et = get_sexp_type se in
if check_formal_actual_call ft et then se else raise (Failure "illegal actual argument found " string_of_typ et " expected " string_of_typ ft " in " string_of_sexp e)) fd.formals actuals

and check_formal_actual_call ft et = match ft with
Vector(t1,_) -> (match et with
| _ -> false)
| Matrix(t1,_) -> (match et with
| Matrix(t2,_) -> check_formal_actual_call t1 t2
| _ -> false)
| Float -> et == Int || et == Float
| _ -> ft == et

and check_vec_access_type v = let t = type_of_identifier v in match t with
Vector(_,_) -> ()
| _ -> raise(Failure "cannot perform vector access on variable " v))

and check_mat_access_type v = let t = type_of_identifier v in match t with
Matrix(_,_,_) -> ()
| _ -> raise(Failure "cannot perform matrix access on variable " v))

and get_mat_row_sexp v e = let se = expr_to_sexp e in let t = type_of_identifier v in match t with
Matrix(_,IntLiteral(j)) -> SCall("get_mat_rowi", [SId(v,t); se], Vector(Int,IntLiteral(j)))
| Matrix(Float,IntLiteral(j)) -> SCall("get_mat_rowf", [SId(v,t); se], Vector(Float,IntLiteral(j)))
| _ -> raise(Failure "cannot get row of non-matrix type")

and get_mat_col_sexp v e = let se = expr_to_sexp e in let t = type_of_identifier v in match t with
Matrix(Int,IntLiteral(i),_) -> SCall("get_mat_coli", [SId(v,t); se], Vector(Int,IntLiteral(i)))
| Matrix(Float,IntLiteral(i),_) -> SCall("get_mat_colf", [SId(v,t); se], Vector(Float,IntLiteral(i)))
| _ -> raise(Failure "cannot get row of non-matrix type")

(* only gets type of vector; does not go through whole vector all the time *)
and get_vec_type el = match el with
IntLiteral(_) :: ss -> get_vec_type ss
| FloatLiteral(_) :: _ -> Float
| [] -> Int
| _ -> raise (Failure "vector/matrix literals can only contain float/int literals")

85
and check_vector_types el =
let t = get_vec_type el in
let check_vec_el e = match e with (* this time check the whole
vector and convert *)
    Int_Literal(i) -> if t == Int then SInt_Literal(i) else
    SFloat_Literal(float i)
| Float_Literal(i) -> SFloat_Literal(i)
| _ -> raise (Failure ("vector/matrix literals can only contain
    float/int literals"))
in
SVector_Literal(List.map check_vec_el el, Vector(t, Int_Literal(
    List.length el)))

and check_matrix_types ell =
let check_row_lengths ell = (* check row lengths *)
let length_first_list = List.length (List.hd ell) in
List.iter (fun l -> if ((List.length l) != length_first_list)
then
    raise (Failure ("matrix row lengths must
    be equal")) else ()) ell
in
let get_mat_type ell =
let rec check_row_types el = match el with
    Int_Literal(_) :: ss -> check_row_types ss
| Float_Literal(_) :: _ -> Float_Literal(0.0)
| [] -> Int_Literal(0)
| _ -> raise (Failure ("vector/matrix literals can only
    contain float/int literals"))
in
get_vec_type (List.map check_row_types ell)
in
let t = get_mat_type ell in
let mat_row_to_smat_row el =
    Int_Literal(i) -> if t == Int then SInt_Literal(i) else
    SFloat_Literal(float i)
| Float_Literal(f) -> SFloat_Literal(f)
| _ -> raise (Failure ("vector/matrix literals can only
    contain float/int literals"))
in
List.map mat_row_to_smat_row el
in
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)))

and compare_vector_matrix_type v1 v2 =
match v1 with
    | Vector(ty1, _) ->
        ( match v2 with
          Vector(ty2, _) -> ty1 == ty2 || ((ty1 == Float) && (ty2 == Int))
          | _ -> raise (Failure ("cannot compare vectors and
              matrices")) )
    | Matrix(ty1, _, _) ->
        ( match v2 with
          Matrix(ty2, _, _) ->
            ty1 == ty2 || ((ty1 == Float) && (ty2 == Int))
          | _ -> raise (Failure ("cannot compare vectors and
              matrices")) )
    | _ -> raise (Failure ("matrix and vector dimensions must be int
              literals"))

and get_binop_sexpr e1 e2 op =
  let se1 = expr_to Sexpr e1 in
  let se2 = expr_to Sexpr e2 in
  match op with
    | Equal | Neq -> get_equality_type se1 se2 op
    | And | Or -> get_binop_boolean_sexpr se1 se2 op
    | Less | Leq | Greater | Geq -> get_binop_comparison_sexpr se1 se2 op
    | Shiftleft | Shiftright | Bitand | Bitor | Bitxor ->
      get_binop_bitwise_sexpr se1 se2 op
    | Add | Sub | Mult | Div | Mod -> get_binop_arithmetic_sexpr se1 se2 op
  in

and get_unop_sexpr op e =
  let se = expr_to Sexpr e in
  match op with
    | Neg -> get_unop_arithmetic_sexpr se op
    | Not -> get_unop_boolean_sexpr se op
    | IntCast | FloatCast -> get_unop_cast_sexpr se op
  in

and get_assign_sexpr e1 e2 =
  let sel = match e1 with
    | Id (_) | VecAccess (_,_) | MatAccess(_,_,_) -> expr_to Sexpr e1
    | _ -> raise (Failure("can only assign to variable or vector/matrix element"))
  in
  let se2 = expr_to Sexpr e2 in
  let lt = get_sexpr_type sel in
  let rt = get_sexpr_type se2 in
  match lt with
    | Vector(_,_) -> if compare_vector_matrix_type lt rt then SAssign (se1,se2,lt) else raise (Failure("illegal assignment "))
    | Matrix(Int,_,_) -> if compare_vector_matrix_type lt rt then SAssign (se1,se2,lt) else raise (Failure("illegal assignment "))
    | Matrix(Float,_,_) -> if compare_vector_matrix_type lt rt then (match rt with
      | Matrix(Int,_,_) -> SAssign (se1,expr_to Sexpr (Unop(FloatCast , e2)),lt) |
      | _ -> (SSAssign (se1,se2,lt)))
      else raise (Failure("illegal assignment "))
    | _ -> if lt == Float && rt == Int then SAssign (se1, (expr_to Sexpr (Unop(FloatCast , e2))), Float)
      else (if lt == rt then SAssign (se1,se2,lt)
        else raise (Failure("illegal assignment " " string_of_typ lt " = " string_of_typ rt ",
        " in: " string_of_expr el " = " string_of_expr e2)))
  in

and check_bool_expr e = if get_sexpr_type (expr_to Sexpr e) != Bool then raise (Failure("expected boolean expression in " string_of_expr e))
else ()

and check_int_expr e = if get_sexpr_type (expr_to Sexpr e) != Int then raise (Failure("expected integer expression in " string_of_expr e))
else ()

and check_int_literal_expr e = match e with
  | Int_Literal(_) -> ()
| . -> raise(Failure("can only declare vectors/matrices/images with int literals"))

let rec stmt_to_sstmt = function

Block(sl) -> let rec check_block = function (* just check if return statement is end of block *) [Return .] -> ()
| Return . -> raise (Failure "nothing may follow a return ")
| Block sl -> check_block (sl @ ss) [ ] -> ()

in

check_block sl; SBlock(List.map stmt_to_sstmt sl)
| Expr(e) -> SExpr(expr_to_sexpr e)
| Return(e) -> let se = expr_to_sexpr e in
if check_formal_actual_call t func.typ then SReturn(se) else
raise (Failure ("return gives " ^ string_of_typ t ^ " expected ".
string_of_typ func.typ ^ " in " ^
string_of_expr e))
| For(e1, e2, e3, st) -> SFor(expr_to_sexpr e1, expr_to_sexpr e2, expr_to_sexpr e3, stmt_to_sstmt st)
| While(p, s) -> check_bool_expr p; SWhile(expr_to_sexpr p, stmt_to_sstmt s)
| Break -> SBreak
| Continue -> SContinue *)

(* check variable declaration type *)
let check_var_decl (t, id) = match t with
Int | Bool | Float | Char | String -> (t, id)
| Vector(t1, e) -> check_int_literal_expr e;
if (t1 != Float) && (t1 != Int) then raise(Failure("can only have vectors/matrices of ints/floats"))
else (); (t, id)
| Matrix(t1, e1, e2) -> check_int_literal_expr e1; check_int_literal_expr e2;
if (t1 != Float) && (t1 != Int) then raise(Failure("can only have vectors/matrices of ints/floats"))
else (); (t, id)
| Image(h, w) -> check_int_literal_expr h; check_int_literal_expr w; (t, id)*

let check_formal_bind (t, id) = match t with
Vector(_,e) -> if e != Noexpr then check_int_literal_expr e else (); (t, id)
| Matrix(_,e1,e2) -> if e1 != Noexpr || e2 != Noexpr then (
check_int_literal_expr e1; check_int_literal_expr e2) else (); (t, id)
| _ -> (t, id)

{
  styp = func.typ;
sfname = func.fname;
 sformals = List.map check_formal_bind func.formals;
 slocals = List.map check_var_decl func.locals;
 sbody = List.map stmt_to_sstmt func.body;
 }

 in
 let sfdecls = List.map fdecl_to_sfdecl functions in
 (globals, sfdecls)
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;
for (i = 0; i < 3; i = i+1) {
    for (j = 0; j < 3; j = j+1) {
        b[i][j] = $int a[i][j];
    }
}
return b;

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

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

    int i;
    int [3] c;
    for (i = 0; i < 3; i = i+1) {
        c[i] = b[i] * a;
    }
    return c;
}

    int i;
    float [3] c;
    for (i = 0; i < 3; i = i+1) {
        c[i] = b[i] * a;
    }
    return c;
}

def int [2][2] scalar_mult_mat2i(int a, int [2][2] b) {
    int i;
    int j;
    int [2][2] c;
    for (i = 0; i < 2; i = i+1) {
        for (j = 0; j < 2; j = j+1) {
            c[i][j] = b[i][j] * a;
        }
    }
    return c;
}

def float [2][2] scalar_mult_mat2f(float a, float [2][2] b) {
    int i;
    int j;
    float [2][2] c;
    for (i = 0; i < 2; i = i+1) {
        for (j = 0; j < 2; j = j+1) {
            c[i][j] = b[i][j] * a;
        }
    }
    return c;
def int[3][3] scalar_mult_mat3i(int a, int[3][3] b) {
    int i;
    int j;
    int[3][3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i][j] = b[i][j] * a;
        }
    }
    return c;
}

def float[3][3] scalar_mult_mat3f(float a, float[3][3] b) {
    int i;
    int j;
    float[3][3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i][j] = b[i][j] * a;
        }
    }
    return c;
}

    int sum;
    int i;
    sum = 0;
    for(i = 0; i < 3; i = i+1) {
        sum = sum + (a[i] * b[i]);
    }
    return sum;
}

    float sum;
    int i;
    sum = 0.0;
    for(i = 0; i < 3; i = i+1) {
        sum = sum + (a[i] * b[i]);
    }
    return sum;
}

    int i;
    int[3] c;
    for(i = 0; i < 3; i = i+1) {
        c[i] = a[i] + b[i];
    }
    return c;
}

    int i;
    float[3] c;
    for(i = 0; i < 3; i = i+1) {

    c[i] = a[i] + b[i];
    }
    return c;
  }

  def int[3][3] mat_mat_add(int[3][3] a, int[3][3] b) {
    int i;
    int j;
    int[3][3] c;
    for (i = 0; i < 3; i = i+1) {
      for (j = 0; j < 3; j = j+1) {
        c[i][j] = a[i][j] + b[i][j];
      }
    }
    return c;
  }

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

    int i;
    int[3] c;
    for (i = 0; i < 3; i = i+1) {
      c[i] = a[i] - b[i];
    }
    return c;
  }

    int i;
    float[3] c;
    for (i = 0; i < 3; i = i+1) {
      c[i] = a[i] - b[i];
    }
    return c;
  }

  def int[3][3] mat_mat_subi(int[3][3] a, int[3][3] b) {
    int i;
    int j;
    int[3][3] c;
    for (i = 0; i < 3; i = i+1) {
      for (j = 0; j < 3; j = j+1) {
        c[i][j] = a[i][j] - b[i][j];
      }
    }
    return c;
  }

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

    int i;
    int j;
    int[3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i] = c[i] + a[j] * b[j][i];
        }
    }
    return c;
}

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

    int i;
    int j;
    int[3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i] = c[i] + a[i][j] * b[j];
        }
    }
    return c;
}

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

def int[3][3] mat_mat_multi(int[3][3] a, int[3][3] b) {
    int i;
    int j;
    int[3][3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j] + a[i][3] * b[3][j];
        }
    }
    return c;
}

    float[3][3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i][j] = c[i][j] + a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j] + a[i][3] * b[3][j];
        }
    }
    return c;
}
return c;
}
    int i;
    int j;
    float [3][3] c;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            c[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j];
        }
    }
    return c;
}
def int [3][3] mat_transposei(int [3][3] a) {
    int i;
    int j;
    int [3][3] b;
    for(i = 0; i < 3; i = i+1) {
        for(j = 0; j < 3; j = j+1) {
            b[j][i] = a[i][j];
        }
    }
    return b;
}
def float [3][3] mat_transposef(float [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[j][i] = a[i][j];
        }
    }
    return b;
}
def float det_mat2(float [2][2] a) {
    return a[0][0] * a[1][1] - a[1][0] * a[0][1];
}
def float det_mat3(float [3][3] a) {
    float [2][2] b;
    float [2][2] c;
    float [2][2] d;
    b[0][0] = a[1][1];
    b[0][1] = a[1][2];
    b[1][0] = a[2][1];
    b[1][1] = a[2][2];
    c[0][0] = a[1][0];
    c[0][1] = a[1][2];
    c[1][0] = a[2][0];
    c[1][1] = a[2][2];
    d[0][0] = a[1][0];
    d[0][1] = a[1][2];
    return a[0][0] * a[1][1] - a[1][0] * a[0][1];
}
```python
d[0][0] = a[1][1];
d[0][1] = a[2][0];
d[1][0] = a[2][1];
return a[0][0] * det_mat2(b) + a[0][1] * det_mat2(c) + a[0][2] * det_mat2(d);
```
```python
def float[2][2] mat_inverse2(float[2][2] a) {
    float[2][2] b;
    b[0][0] = a[1][1];
    b[0][1] = -a[0][1];
    b[0][0] = -a[1][0];
    return (1 / det_mat2(a)) * b;
}
```
```python
def float[3][3] mat_inverse3(float[3][3] a) {
    float[2][2] b11;
    float[2][2] b12;
    float[2][2] b13;
    float[2][2] b21;
    float[2][2] b22;
    float[2][2] b23;
    float[2][2] b31;
    float[2][2] b32;
    float[2][2] b33;
    float[3][3] c;
    b11[0][0] = a[1][1];
    b11[0][1] = a[1][2];
    b11[1][0] = a[2][1];
    b11[1][1] = a[2][2];
    b12[0][0] = a[0][2];
    b12[0][1] = a[0][1];
    b12[1][0] = a[2][2];
    b12[1][1] = a[2][1];
    b13[0][0] = a[0][1];
    b13[0][1] = a[0][2];
    b13[1][0] = a[1][1];
    b13[1][1] = a[1][2];
    b21[0][0] = a[1][2];
    b21[0][1] = a[1][0];
    b21[1][0] = a[2][2];
    b21[1][1] = a[2][0];
    b22[0][0] = a[0][0];
    b22[0][1] = a[0][2];
    b22[1][0] = a[2][0];
    b22[1][1] = a[2][2];
    b23[0][0] = a[0][2];
    b23[0][1] = a[0][0];
    b23[1][0] = a[1][2];
    b23[1][1] = a[1][0];
    b31[0][0] = a[1][0];
    b31[0][1] = a[1][1];
    b31[1][0] = a[2][0];
```
b31[1][1] = a[2][1];
b32[0][0] = a[0][1];
b32[0][1] = a[0][0];
b32[1][0] = a[2][1];
b32[1][1] = a[2][0];
b33[0][0] = a[0][0];
b33[0][1] = a[0][1];
b33[1][0] = a[1][0];
b33[1][1] = a[1][1];
c[0][0] = det_mat2(b11);
c[0][1] = det_mat2(b12);
c[0][2] = det_mat2(b13);
c[1][0] = det_mat2(b21);
c[1][1] = det_mat2(b22);
c[1][2] = det_mat2(b23);
c[2][0] = det_mat2(b31);
c[2][1] = det_mat2(b32);
c[2][2] = det_mat2(b33);
return (1 / det_mat3(a)) * c;
}
def int[3] get_mat_rowi(int[3][3] a, int b) {
    int[3] ret;
    int i;
    for (i = 0; i < 3; i = i + 1) {
        ret[i] = a[b][i];
    }
    return ret;
}
def float[3] get_mat_rowf(float[3][3] a, int b) {
    float[3] ret;
    int i;
    for (i = 0; i < 3; i = i + 1) {
        ret[i] = a[b][i];
    }
    return ret;
}
def int[3] get_mat_coli(int[3][3] a, int b) {
    int[3] ret;
    int i;
    for (i = 0; i < 3; i = i + 1) {
        ret[i] = a[i][b];
    }
    return ret;
}
def float[3] get_mat_colf(float[3][3] a, int b) {
    float[3] ret;
    int i;
    for (i = 0; i < 3; i = i + 1) {
        ret[i] = a[i][b];
    }
    return ret;
}