Matrix Language (ML)
Final Report
COMS W4115

Alex Barkume • Jared Greene • Kyle Jackson • Caroline Trimble • Jessica Valarezo

May 11, 2016

Table of Contents

1. An Introduction to the Language

ML is a programming language with an emphasis on matrix manipulation, which is specifically applicable to image processing, as images can be represented as large matrices of pixels in the ppm format. This ability is particularly applicable to applying computer vision algorithms to images, including filtering, transformations, color space conversions, thresholding, and various others. Although the main objective for ML is to provide an imperative language specifically designed for image processing, ML provides an extensive standard library specifically designed for matrix manipulation and, as such, also proves exceptionally useful for complex matrix manipulation outside the realm of image processing.

In regard to image processing, ML will support the P3 ppm simple image file type, which represents images as a text file using ASCII characters to represent pixels. Upon attempting to open a file, the ML compiler automatically assumes that this is a P3 ppm image file, and creates a matrix data type to represent that image file. Thus, after opening a file and assigning it to a variable, a user can simply treat the image variable as a matrix type. Once given this matrix data type of tuples that represents an image, the user can write a wide-variety of functions that manipulate the rows and/or columns of the matrix by iterating over them.

To utilize the PPM image format, a user must download GIMP - https://www.gimp.org/downloads/ - After downloading GIMP, a user can import any image file type (PNG, JPG, etc.) into GIMP and then export it as a ASCII-type PPM. Once a user has manipulated the image using ML, he can also open and view it using GIMP. For more information on GIMP and the PPM image file format, please refer to https://www.gimp.org/about/ and http://netpbm.sourceforge.net/doc/ppm.html , respectively.

ML compiles to the Low Level Virtual Machine

2. Language Tutorial

ML makes it easy to create and manipulate matrices. Before we begin the tutorial, you should know the two data types supported in ML.

2.1 Data Types

There are two basic data types, aside from primitive types, defined by ML. The primitive types range from int, float, bool and char. Type identifiers will always begin with any letter, and thereafter, an identifier can have any combination of lowercase and uppercase letter, numbers, and underscore.

- `Tuple`: A list of int, char or float, and must all be the same type.
- `Matrix`: A list of ints, floats or tuples, where the tuples are a list of ints or floats. A tuple matrix will have tuples that are all one type and length.

2.2 Basic Syntax

```
// Declaration must precede assignment
int i;
int d;
int[3] pixel;
int[3][2:2] m1;
int[3:3] m2;

i=0;
// These both assign a pixel p with 0's
      pixel = [|0,0,0|];
      pixel = [|i,i,i|];
      // Assign an int matrix
      m1 = [|1,1,1|1,1,1|1,1,1|];
      // Assign an int tuple matrix
      m2 = [|(0,0,0),(0,0,0)|(0,0,0),(0,0,0)|];
      // Use method from standard library which returns determinant
      d = detThree(@@m1);
      // Use method from standard library which updates matrix
      // uses # of rows, columns and tuple length of tuple matrix
      bwIntTupleMatrix(@@@m2, m2.rows, m2.columns, m2.length);
```

2.3 Program Structure

Programs written in ML must define a main method with the following declaration:

```
int main()
```

The main method can call other user-defined methods, which may be recursive. In addition to any local variables defined within the scope of a specific method, the user can also define global variables outside the scope of any method. These global variables can be accessed by any of the program's methods. To access the standard library, or any other .mxl files, the user should include it at the beginning of the file, with the following declaration:

```
#include <stdlib.mxl>;
```

2.4 Pointers, Accessing, Updating Elements

A particularly neat aspect of our language is the ability to access the pointer to the first element of a tuple, matrix, or tuple matrix, using `@pixel`, `@@m1`, and `@@@m2`. You can pass the pointer around as a function argument, making it possible to update values in memory from within a method, which will persist after. It is also possible to dereference a pointer, using `$id`, to use the value it points to. Lastly, to increment or decrement a pointer, use `id.+.` It is possible assign a pointer to a variable:

```
int[][][] pval;
pval = m1;
```

An element in a two-dimensional matrix is accessed by using the subscripts, i.e. row index and column index of the matrix. If saving in a variable, the type of the variable must match the type of the element (i.e. the type of the matrix).

Updating the values of a matrix or tuple can be done using the pointer, or simply by reassigning:

```
m1[1]=1;
m2[0:0] = pixel;
```

2.5 Examples

2.5.1 Matrix Arithmetic

```
void addIntMatrices(int[][] x, int[][] y, int r, int c) {
    int len;
    int i;

    len = r * c;

    for(i = 0; i < len; i = i + 1) {
        $x = $x + $y;
        x = x.+;
               y = y.+;
           }
       }

       int main() {
           int[2:2] a;
           int[2:2] b;
           int i;
           int j;

           a = [| 1, 2 | 3, 4 | ];
           b = [| 5, 7 | 9, 15| ];

           addIntMatrices(@@a, @@b, a.rows, a.columns);

           for(i = 0; i < a.rows; i = i + 1) {
               for (j = 0; j < a.columns; j = j + 1) {
                   print(a[i:j]);
               }
           }

           return 0;
       }
```

2.6 Compile

To compile your .ml files, first **make** the compiler. You will now have file `ml.native`

Now, to actually compile your first project, execute the command

```
./ml.native file_namp.ml > temp_file.ll
Llc -filetype=obj test.ll
g++ test.o
```

For this, you will need the LLVM compiler, LLC. For more information about this and instructions on how to obtain LLC for your personal computer, turn to the README file in the folder ml-llvm.

3. Language Reference Manual

3.1.Types

3.1.1 Primitive Types

Type	Description
`int`	32-bit signed integer value. Can hold values ranging from -2,147,483,648 to 2,147,483,647, inclusive.
`float`	64-bit signed floating point value. The float type can hold values ranging from 1e-37 to 1e37.
`bool`	1-bit Boolean value: true or false
`char`	8-bit character value. A char value maps to an ASCII code. The decimal values 0 through 31, and 127, represent non-printable control characters. All other characters are printable. The character ‘A’ has the code value of 65, ‘B’ has the value 66, and so on. The ASCII values of letters ‘A’ through ‘Z’ are in a contiguous increasing numeric sequence. Similarly, the lowercase letters, ‘a’ through ‘z’ are stored contiguously starting at value 97, while the digit symbol characters ‘0’ through ‘9’ start at value 48.

3.1.2 Data Types

Type	Description
`tuple`	A constant array of immutable length stored contiguously in memory. Elements can be int, char, bool, or float. Elements must all be the same type.
`matrix`	A constant multidimensional array of immutable length stored contiguously in memory. Matrix of tuple, int, or float. Every tuple must be of type int or float, and each tuple must be of the same length. All elements of a matrix must be of the same type.

3.1.3 Pointers

ML supports various types of pointers. The intended use of these pointers is to pass the data types tuple and matrix by reference into user defined and standard library functions. The access, dereferencing, and incrementing of these pointers is discussed in the *Operators* section of this report. Assignment of these pointers to variables is covered in the *Tutorial* section of this report.

Type	**Description**
`int[]`	Pointer to the first element of an `int` tuple.
`int[][]`	Pointer to the first element of an `int` matrix.
`int[][][]`	Pointer to the first element of a matrix of `int` tuples.
`float[]`	Pointer to the first element of a `float` tuple.
`float[][]`	Pointer to the first element of a `float` matrix.
`float[][][]`	Pointer to the first element of a matrix of `float` tuples.
`bool[]`	Pointer to the first element of a `bool` tuple.
`char[]`	Pointer to the first element of a `char` tuple.

3.1 Lexical Conventions

3.1.1 Identifiers

An identifier is a sequence of characters utilized for naming a primitive type, data type, or function. An identifier can begin with any lowercase or uppercase letter (^[A-Za-z]). Thereafter, an identifier can have any combination of lowercase and uppercase letters, numbers, and underscore ([0-9A-Za-z_]+$). An identifier can not have the same character sequence as any reserved word in ML, such as `if, tuple,` or `float.` Identifiers are case sensitive and thus `value` and `vALUE` are two separate identifiers.

3.1.2 Reserved Words

A reserved word is a special identifier reserved for use as part of the programming language itself and thus you can not use it for any other purpose. ML recognizes the following reserved words:

Reserved Word	**Description**
`#include <file_name>`	Imports declarations and implementations from file_name during compile time. Include statements must be the first lines in the program
`open`	Used to open a ppm file that will be declared as a matrix by the compiler. Use: `variable_name = open("filename.ppm");` Covered in detail in the I/O section of this report.
`for`	*for(expr1; expr2; expr3) statement* where expr2 is a relational expression.
`if`	If in if-else statement. *if (expr1) statement* where expr1 is a relational expression
`else`	Else in if-else statement. *else statement*
`while`	*while(expr) statement* where expr is a relational expression
`for`	*for(expr1; expr2; expr3) statement* where expr2 is a relational expression.
`main`	Main function. The code within the main function will be executed when the executable runs. The main function always returns type int.
`return`	Return function value. In the case that the function has spawned threads, waits for all threads to terminate.
`true`	Boolean literal value (True).
`false`	Boolean literal value (False).
`void`	No type
`free`	Used to free a heap allocated matrix
`length`	Returns the length of a tuple or the length of the tuples in a tuple matrix. Use: `variable_name.length`
`rows`	Returns the number of rows in a matrix. Use: `variable_name.rows`
`columns`	Returns the number of columns in a matrix. Use: `variable_name.columns`

3.1.3 Comments

```
// Single line comment

/**
Block comment
spans multiple lines
**/
```

3.1.4 Operators

Operators	Description
=	Assignment operator. Note: the left-hand side and the right-hand side of the assignment operator must be of the same data type.
*	Multiplication Operator. Uses: `int * int, float * float.` Only used with ints and floats; types of two operands must match. There must be exactly two operands.
/	Division Operator. Uses: `int / int, float / float` Only used with ints and floats; types of two operands must match. There must be exactly two operands.
+	Addition Operator. Uses: `int + int, float + float` Only used with ints and floats; types of two operands must match. There must be exactly two operands.
-	Subtraction Operator. Uses: `int - int, float - float.` Only used with ints and floats; types of two operands must match. There must be exactly two operands.
<	Less than comparison. Uses: `int < int, float < float.` Only used with ints and floats; types of two operands must match. There must be exactly two operands. either a 1 or 0 for true or false, respectively.
>	Greater than comparison. Uses: `int > int, float > float.` Only used with ints and floats; types of two operands must match. There must be exactly two operands. Returns either a 1 or 0 for true or false, respectively.
>=	Greater than or equal to comparison. Uses: `int >= int, float >= float`. Only used with ints and floats; types of two operands must match. There must be exactly two operands. Returns either a 1 or 0 for true or false, respectively.
<=	Less than or equal to comparison. Uses: `int <= int, float <= float.` Only used with ints and floats; types of two operands must match. There must be exactly two operands. Returns either a 1 or 0 for true or false, respectively.

`==`	Equality than or equal to comparison. Uses: `int == int, float == float, bool == bool.` Only used with ints, floats, and bools; types of two operands must match. There must be exactly two operands. Returns either a 1 or 0 for true or false, respectively.
`!=`	Inequality comparison. Uses: `int != int, float != float, bool != bool.` Only used with ints, floats, and bools; types of two operands must match. There must be exactly two operands. Returns either a 1 or 0 for true or false, respectively.
`&&`	Logical AND operator. Uses: `bool && bool`, `Boolean expression && Boolean Expression.` Type of both operands must be bool or Boolean expression. Returns either a 1 or 0 for true or false, respectively.
`||`	Logical OR operator. Uses: `bool || bool`, `Boolean expression || Boolean Expression.` Type of both operands must be bool or Boolean expression. Returns either a 1 or 0 for true or false, respectively.
`!`	Logical NOT operator. Uses `!bool`, `!Boolean expression.` Only used with bool primitives and expressions that evaluate to a bool type. Must be exactly one operand.
`;`	Statement separator.
`[]`	Access an element of a tuple or matrix. Use: `identifier[expr].` Type of `identifier` can only be tuple or matrix. The expression within the tuple access must be of data type int.
@	Accesses the pointer to the first element of a tuple. Use: `@identifier`. Type of `identifier` can only be a tuple. Used for passing in a tuple to standard library or user defined tuple functions. @ is equivalent to datatype[].
@@	Accesses the pointer to the first element of a matrix. Use: `@@identifier.` Type of `identifier` can only be a matrix. Used for passing in a matrix to standard library or user defined matrix functions. @@ is equivalent to datatype[][][]
@@@	Accesses the pointer to the first element of the first tuple in a matrix of tuple. Use: `a@@identifier.` Type of identifier may only be a matrix of tuples. Used for passing in a matrix to standard library or user defined matrix functions.
`$`	Dereference a tuple, matrix, or matrix tuple pointer. Use: `$identifier` or `$identifier = expression`. If assigning to a variable, as in `variable_name = $identifier`, then the type of variable_name must match the primitive type of the `identifier.`
`.+`	Increment a pointer. Use: `identifier.+.` There are no explicit checks to see if the user incremented the pointer too far.

3.1.5 Literals

A literal is the source code representation of the value of a primitive type.

3.1.5.1 Integer Literals

An integer literal is expressed in decimal (base 10). It is represented with either the single ASCII digit 0, representing the integer zero, or an ASCII digit from 1 to 9 optionally followed by one or more ASCII digits from 0 to 9. That is, an integer can be expressed by `['0'-'9']+.` An integer literal is not surrounded by any form of quotation marks.

3.1.5.2 Float Literals

A float literal is made up of an integer part, a decimal part (represented by the ASCII period), and a fraction part. The integer and fraction parts are defined by a single digit 0 or one digit from 1-9 followed by more ASCII digits from 0 to 9. That is, a float can be expressed `by ['0'-'9']+ ['.'] ['0'-'9']+`. A float literal is not surrounded by any form of quotation marks.

3.1.5.3 Bool Literals

A bool literal is represented by ASCII characters. A bool literal is either `true` or `false`. A bool literal is not surrounded by any form of quotation marks.

3.1.5.3 String Literals

Although ML does not support a string type, a user can define a string literal (e.g. to be used in a print statement). String literals can be made up of one or more ASCII characters surrounded by double quotation marks.

3.2. Syntax

All statements in ML must end with a semi-colon (`;`). Additionally, code blocks under a loop or function must be enclosed in brackets if there are multiple statements within the block.

3.2.1 Expressions

Assignment Expressions
The assignment operators are binary operators with right-to-left associativity. It is an identifier and an expression separated by an equals sign.

Arithmetic Expressions
The arithmetic operator represents basic mathematical operations with left-to-right associativity.

Function Calls
To be able to call a function, it must have been declared and implemented previously. The function call will execute using the given arguments and return whatever value was defined as the return type during its declaration.

If the argument passed in to the function is a matrix or tuple, it is passed by referenced and when called must use @ for tuples or @@ for matrices to indicate a pointer. Because arguments are passed by reference, functions can be built to have side effects.

If the argument passed into the function is a primitive type, it will be passed by value.

The function call will return the value of the data type defined as a return type in the function declaration.

```
    function_name(<arguments>);
```

3.2.2 Declaration and Initialization

Primitive Type Declaration

Basic data types are declared in the format:

```
    type variable_name;
```

For example:

```
int x;
bool b;
```

Variables can be local, formal arguments in a function, or global. The precedence of these variables is as follows:

Precedence	Variable Type
highest	local
	formal
lowest	global

All variable declarations must occur in the first lines of the program, before any assignment or function calls take place.

Primitive Type Initialization

Basic data types must be initialized in two separate lines of the program. The data type must first be declared and then it can be initialized in any subsequent line, as follows:

```
    type variable_name;
    variable_name = literal;
```

For example:

```
int x;
x = 7;
```

```
bool y;
y = true;
```

Tuple Declaration

Tuples are declared in the following format:

```
primitive_type[length] tuple_name;
```

Where `primitive_type` is either int, float, bool, or char and length is an integer value greater than or equal to 1.

All variable declarations must occur in the first lines of the program, before any assignment or function calls take place.

Tuple Initialization

Tuples must be declared and initialized on two separate lines of the program. The tuple must first be declared and then it can be initialized in any subsequent line, as follows:

```
tuple_name = [literal, literal, literal,...];
```

Where there are **exactly** as many values in the declared tuple as the stated length in the declaration. As previously mentioned, a tuple can be of any primitive type (i.e. one can not "nest" tuples, and a tuple can not hold a matrix). Furthermore, every element in the tuple must be of the same primitive type.

Matrix Declaration

Matrices must be declared in the following format:

```
primitive_type[len_row:len_col] matrix_name;
```

where row and column are integers.

All variable declarations must occur in the first lines of the program, before any assignment or function calls take place.

Example:

```
int[2:2] m1;
```

Matrices that contain tuples must be declared in the following format:

```
primitive_type[len_tuple][len_row:len_column] matrix_name;
```

Matrix Initialization

Similar to primitive types and tuples, matrices must be declared first and then initialized on a subsequent line. It is best to display matrix declaration through example.

For the primitive type matrix:

```
a  b  c

d  e  f

g  h  i
```

The declaration is as follows:

```
     matrix_name = [|a, b, c| d, e, f| h, i, j|]*
```

For the tuple matrix:

```
(a,b,c) (d,e,f)

(h,i,j) (l,m,n)
```

The declaration is as follows:

```
     matrix_name = [|(a,b,c), (d,e,f)|(h,i,j), (l,m,n)|]*
```

* a, b, c are primitive literals

Accessing a Tuple Element

An element in a tuple is accessed by using its index in brackets following the tuple variable name. If saving in a variable, the type of the variable must match the type of the element (i.e. the type of the tuple).

```
     type variable_name = tuple_name[i];
```

Accessing a Matrix Element

An element in a two-dimensional matrix is accessed by using the subscripts, i.e. row index and column index of the matrix. If saving in a variable, the type of the variable must match the type of the element (i.e. the type of the matrix).

```
     type variable_name = matrix_name[row][column];
```

Example:

```
     int val = m0[2][3];
```

Function Declaration

Functions consist of a function header and a function body. The function header contains the return type, which can be any of the data types, including void. A function that has no return type and is being used solely for its side effect must be declared as type void. The header also contains the function name, which must be a valid identifier, and an optional parameter list enclosed in parentheses. The function body is enclosed in curly braces. A function may not be overloaded, that is, all functions must have a **unique** name.

A function can return the following types:

- `int`
- `float`
- `char`
- `bool`
- `int[]`
- `float[]`
- `char[]`
- `bool[]`
- `int[][]`
- `float[][]`
- `int[][][]`
- `float[][][]`

```
// Function with a return type
return_type <function_name> (<parameters>){ statement }
```

or it can return nothing (void):

```
// Function with no return type
void <function_name> (<parameters>){ statement }
```

3.3 Control Flow

3.3.1 Conditional Statements

The conditional statements in ML are formed by the reserved words `if` and `else`.
They can be of the form:
`if``(`*`expression`*`)` *`statement;`*
or
`if``(`*`expression`*`)` *`statement`* **`else`** *`statement;`*

The expression enclosed in balanced parentheses is evaluated and if it is true, the first sub-statement is executed. In the first form, where there is no **else** statement, the program just continues on to the subsequent lines if the expression evaluates to false. However, like in the second case, if there is an **else** statement, and the first expression evaluates to false, the statement following the **else** reserved word is executed. Ambiguity regarding **else** is resolved by connecting an else with the last encountered else-less if.

3.1.2 Looping Structures

`while``(`*`expression`*`)` *`statement`*

The statement is executed repeatedly so long as the value of the expression evaluates to non-zero result. The test takes place before each execution of the statement.

`for`*`(expression; expression; expression) statement`*

The first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the expression evaluates to 0; the third expression typically specifies an increment (on the variable from the first initializing expression) which is performed after each iteration.

`return`
The return statement is used within the scope of a function:

```
    return expression;
```

In the case of a void function:

```
    return;
```

3.4 Input / Output

3.4.1 Input

Users can open a P3 ppm file using the statement:

```
    open(file_name.ppm)
```

Which will return a matrix. To use this matrix in the program, a user will simply need to assign the file open to a variable name.

```
    m1 = open(file_name.ppm)
```

After a user “catches” a file-open in a variable name, he can use this variable as he would any other matrix-type variable and execute any operations and functions that are applicable to matrices (access, element assignment, etc.)

3.4.2 Output

There are various print statements available in ML.

```
print:
     print(variable_name)
     print(//integer literal)
```

The `print` statement can be used to print `int` literals and variables

```
printb
     printb(//bool literal)
     printb(//bool variable)
```

The `printb` statement can be used to print bool literals and variables.

```
printf:
     printf(variable_name)
     printf(//float literal)
```

The `printf` statement can be used to print `float` variables and literals.

```
printc
     printc(//char literal)
     printc(//char variable)
```

The `printc` statement can be used to print char literals and variables.

```
prints
     prints(//string literal)
```

The `prints` statement can be used to print string literals.

```
printsl
     printsl(//int literal)
     printsl(//int variable)
```

The `printsl` statement can be used to print int literals and variables and it **does not print a new line.**

```
printbsl
     printbsl(//int literal)
     printbsl(//int variable)
```

The `printbsl` statement can be used to print bool literals and variables and it **does not print a new line.**

```
printfsl
     printfsl(//float literal)
     printfsl(//float variable)
```

The `printfsl` statement can be used to print float literals and variables and it **does not print a new line.**

```
printcsl
     printcsl(//char literal)
     printcsl(//char variable)
```

The `printcsl` statement can be used to print char literals and variables and it **does not print a new line.**

```
printssl
     printssl(//string literal)
```

The `printssl` statement can be used to print string literals and it **does not print a new line.**

3.5 Exceptions

Binop Exceptions

```
exception UnsupportedBinaryOperationOnTypes of string * string
exception UnsupportedBinOp
```

These exceptions are called when a user misuses any of the binary operators of ML. Particularly if a user attempts to do a binary operation on strings (e.g `string_variable + string_variable)` the exeption `UnsupportedBinaryOperationOnTypes` will be called. If a user attempts to executed an unsupported binary operation, especially with type mismatch (e.g. `float_variable * int_variable)` the exception `exception UnsupportedBinOp` will be called.

Unop Exceptions

```
exception UnsupportedUnaryOperationOnType of string
exception UnsupportedUnaryOperationOnFloat
```

These exceptions are called when a user attempts to use a unary operator on an unsupported type variable or literal, such as a string literal.

Tuple Exceptions

```
exception UnsupportedTupleOfTuples
exception UnsupportedTupleType
exception InvalidTuplePointerType
exception InvalidUseOfLength
```

These exceptions are called when a user attempts to utilize the tuple data type in an unsupported way, such as creating a tuple of type tuple, creating a tuple of type matrix. Furthermore, these exceptions will be called if a user misuses the length attribute or tuple pointers.

Matrix Exceptions

```
exception UnsupportedMatrixofMatrices
exception UnsupportedMatrixType
exception InvalidUseOfRows
exception InvalidUseOfColumns
```

These exceptions are called when a user attempts to create a matrix of an unsupported type (e.g. a matrix of type matrix) or misuses the row or column attributes of the matrix data type.

Assignment Exceptions

```
exception IllegalAssignment
```

This exception is called when a user attempts to assign an expression to a variable in an unsupported way (e.g. type mismatch)

Pointer Exceptions

```
exception IllegalPointerType
```

This exception is called when a user attempts to assign an expression to a pointer variable in an unsupported way (e.g. type mismatch)

Return Exceptions

```
exception IllegalReturnType
```

This exception is called when a user attempts to return an unsupported type from a function. The full list of supported types is included in the *Function Declaration* section of this report.

3.5 Standard Library Functions

```
/* NAME: printFloatTuple
   ARGUMENTS: Pointer to float tuple (@variable_name) and tuple length
   EXECUTES: Prints the values of the tuple, each on a new line
   RETURNS: VOID
*/
void printFloatTuple(float[] x,  int len) {
  int i;
  float f;

  printssl("[");

  for (i = 0; i < len; i = i + 1) {
    f = $x;
        printfsl(f);

    if (i != len - 1) {
      printssl(", ");
    }

      x = x.+;
  }

  prints("]");
}
/* NAME: printIntTuple
   ARGUMENTS: Pointer to int tuple (@variable_name) and tuple length
   EXECUTES: Prints the values of the tuple, each on a new line
   RETURNS: VOID
*/
void printIntTuple(int[] x,  int len) {
  int i;
  int f;

  printssl("[");

  for (i = 0; i < len; i = i + 1) {
    f = $x;
        printsl(f);

    if (i != len - 1) {
      printssl(", ");
    }

        x = x.+;
  }

  prints("]");
}

/* NAME: printIntMatrix
   ARGUMENTS: Pointer to int matrix (@@x), number of rows (int), number of columns (int)
   EXECUTES: Prints the values of the contents of the matrix. Each row on a new line and each
             ([|...|]).
   RETURNS: void
*/

void printIntMatrix(int[][] x, int r, int c) {
  int len;
  int i;
  int j;

  len = r * c;

  printssl("[| ");

  for (i = 0; i < r; i = i + 1) {
    for (j = 0; j < c; j = j + 1) {
      if (j != c - 1) {
        printsl($x);
        printssl(", ");
        x = x.+;
      } else {
        printsl($x);
        if (i != r - 1) {
          prints("");
        }
      }
    }
    if (i != r - 1) {
      printssl(" | ");
    }
    x = x.+;
  }

  prints(" |]");
}

/* NAME: printFloatMatrix
   ARGUMENTS: Pointer to int matrix (@@x), number of rows (int), number of columns (int)
   EXECUTES: Prints the values of the contents of the matrix. Each row on a new line and each
              ([|...|]).
   RETURNS: void
*/

void printFloatMatrix(float[][] x, int r, int c) {
  int len;
  int i;
  int j;

          len = r * c;

          printssl("[| ");

          for (i = 0; i < r; i = i + 1) {
            for (j = 0; j < c; j = j + 1) {
              if (j != c - 1) {
```

```
        printfsl($x);
        printssl(", ");
        x = x.+;
      } else {
        printfsl($x);
        if (i != r - 1) {
          prints("");
        }
      }
    }
    if (i != r - 1) {
      printssl(" | ");
    }
    x = x.+;
  }

  prints(" |]");
}



/* NAME: addIntTuples
   ARGUMENTS: Pointer to int tuple (@x), Pointer to int tuple (@y),
              tuple length (tuples must be of the same length, so length of x or y can be passed in)
   EXECUTES: Adds the contents of y to the contents of x; updating x in memory
   RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
*/
void addIntTuples(int[] x, int[] y, int len) {
    int i;

    for (i = 0; i < len; i = i + 1) {
        $x = $x + $y; // $ dereferences the pointer
        x = x.+;      // Incrementing the pointer by 1
        y = y.+;          // Incrementing the pointer by 1
    }
    }


    /* NAME: subtractIntTuples
       ARGUMENTS: Pointer to int tuple (@x), Pointer to int tuple (@y),
                  tuple length (tuples must be of the same length, so length of x or y can
 be passed in)
       EXECUTES: Subtracts the contents of y from the contents of x; updating x in memory
       RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
    */
    void subtractIntTuples(int[] x, int[] y, int len) {
        int i;

        for (i = 0; i < len; i = i + 1) {
            $x = $x - $y; // $ dereferences the pointer
            x = x.+;      // Incrementing the pointer by 1
            y = y.+;          // Incrementing the pointer by 1
        }
    }

    /* NAME: multiplyIntTuples
       ARGUMENTS: Pointer to int tuple (@x), Pointer to int tuple (@y),
                  tuple length (tuples must be of the same length, so length of x or y can
 be passed in)
       EXECUTES: Multiply the contents of y with the contents of x; updating x in memory
       RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
    */
    void multiplyIntTuples(int[] x, int[] y, int len) {
        int i;

        for (i = 0; i < len; i = i + 1) {
            $x = $x * $y; // $ dereferences the pointer
            x = x.+;      // Incrementing the pointer by 1
            y = y.+;          // Incrementing the pointer by 1
        }
    }
```

```

        /* NAME: divideIntTuples
           ARGUMENTS: Pointer to int tuple (@x), Pointer to int tuple (@y),
                      tuple length (tuples must be of the same length, so length of x or y can
be passed in)
           EXECUTES: Divide x by y; updating x in memory and tossing the remainder
           RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
        */
        void divideIntTuples(int[] x, int[] y, int len) {
            int i;

            for (i = 0; i < len; i = i + 1) {
                $x = $x / $y; // $ dereferences the pointer
                x = x.+;      // Incrementing the pointer by 1
                y = y.+;          // Incrementing the pointer by 1
            }
        }


        /* NAME: addFloatTuples
           ARGUMENTS: Pointer to float tuple (@x), Pointer to float tuple (@y),
                      tuple length (tuples must be of the same length, so length of x or y can
be passed in)
           EXECUTES: Add the contents of y to the contents of x; updating x in memory
           RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
        */
        void addFloatTuples(float[] x, float[] y, int len) {
            int i;

            for (i = 0; i < len; i = i + 1) {
                $x = $x + $y; // $ dereferences the pointer
                x = x.+;      // Incrementing the pointer by 1
                y = y.+;          // Incrementing the pointer by 1
            }
        }

```

```
/* NAME: subtractFloatTuples
   ARGUMENTS: Pointer to float tuple (@x), Pointer to float tuple (@y),
              tuple length (tuples must be of the same length, so length of x or y can be passed in)
   EXECUTES: Subtract the contents of y from the contents of x; updating x in memory
   RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
*/
void subtractFloatTuples(float[] x, float[] y, int len) {
    int i;

    for (i = 0; i < len; i = i + 1) {
        $x = $x - $y; // $ dereferences the pointer
        x = x.+;      // Incrementing the pointer by 1
        y = y.+;          // Incrementing the pointer by 1
    }
}


/* NAME: multiplyFloatTuples
   ARGUMENTS: Pointer to float tuple (@x), Pointer to float tuple (@y),
              tuple length (tuples must be of the same length, so length of x or y can be passed in)
   EXECUTES: Multiply the contents of y by the contents of x; updating x in memory
   RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
*/
void multiplyFloatTuples(float[] x, float[] y, int len) {
    int i;

    for (i = 0; i < len; i = i + 1) {
        $x = $x * $y; // $ dereferences the pointer
        x = x.+;      // Incrementing the pointer by 1
        y = y.+;          // Incrementing the pointer by 1
    }
}


```

```
/* NAME: divideFloatTuples
   ARGUMENTS: Pointer to float tuple (@x), Pointer to float tuple (@y),
              tuple length (tuples must be of the same length, so length of x or y can be passed in)
   EXECUTES: Divide the contents of x by the contents of y; updating x in memory; toss remainder
   RETURNS: VOID; the first tuple variable is updated in memory; nothing is returned
*/
void divideFloatTuples(float[] x, float[] y, int len) {
    int i;

    for (i = 0; i < len; i = i + 1) {
        $x = $x / $y; // $ dereferences the pointer
        x = x.+;      // Incrementing the pointer by 1
        y = y.+;          // Incrementing the pointer by 1
    }
}


/* NAME: addIntMatrices
   ARGUMENTS: Pointer to int matrix (@@x), pointer to int matrix (@@y)
               number of rows (int), number of columns (int)
   EXECUTES: Add each element in matrix x with its corresponding element
             in matrix y. Updates x in memory with the sum.
   RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
   NOTE: Both matrices must have the same number of rows and columns. This is why the this are
         only passed in once, because they are expected to be the same.
*/
void addIntMatrices(int[][] x, int[][] y, int r, int c) {
    int len;
    int i;

    len = r * c;

    for(i = 0; i < len; i = i + 1) {
```

```
                $x = $x + $y;
                x = x.+;
                y = y.+;
            }
        }



        /* NAME: subtractIntMatrices
           ARGUMENTS: Pointer to int matrix (@@x), pointer to int matrix (@@y)
                        number of rows (int), number of columns (int)
           EXECUTES: Subtract each element in matrix x with its corresponding element
                       in matrix y. Updates x in memory with the difference.
           RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
           NOTE: Both matrices must have the same number of rows and columns. This is why the
    this are
                  only passed in once, because they are expected to be the same.
        */
        void subtractIntMatrices(int[][] x, int[][] y, int r, int c) {
            int len;
            int i;

            len = r * c;

            for(i = 0; i < len; i = i + 1) {
                $x = $x - $y;
                x = x.+;
                y = y.+;
            }
        }


        /* NAME: multiplyIntMatrices
           ARGUMENTS: Pointer to int matrix (@@x), pointer to int matrix (@@y)
                        number of rows (int), number of columns (int)
           EXECUTES: Multiply  each element in matrix x with its corresponding element
```

```
                 in matrix y. Updates x in memory with the product.
       RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
       NOTE: Both matrices must have the same number of rows and columns. This is why the this are
             only passed in once, because they are expected to be the same.
    */
    void multiplyIntMatrices(int[][] x, int[][] y, int r, int c) {
        int len;
        int i;

        len = r * c;

        for(i = 0; i < len; i = i + 1) {
            $x = $x * $y;
            x = x.+;
            y = y.+;
        }
    }


    /* NAME: divideIntMatrices
       ARGUMENTS: Pointer to int matrix (@@x), pointer to int matrix (@@y)
                   number of rows (int), number of columns (int)
       EXECUTES: Divide  each element in matrix x with its corresponding element
                 in matrix y. Updates x in memory with the quotient. Tosses the remainder.
       RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
       NOTE: Both matrices must have the same number of rows and columns. This is why the this are
             only passed in once, because they are expected to be the same.
    */
    void divideIntMatrices(int[][] x, int[][] y, int r, int c) {
        int len;
        int i;

        len = r * c;

```

```
            for(i = 0; i < len; i = i + 1) {
                $x = $x / $y;
                x = x.+;
                y = y.+;
            }
        }

        /* NAME: addFloatMatrices
           ARGUMENTS: Pointer to float matrix (@@x), pointer to float matrix (@@y)
                      number of rows (int), number of columns (int)
           EXECUTES: Add each element in matrix x with its corresponding element
                      in matrix y. Updates x in memory with the sum.
           RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
           NOTE: Both tuples must have the same number of rows and columns. This is why the
this are
                  only passed in once, because they are expected to be the same.
        */
        void addFloatMatrices(float[][] x, float[][] y, int r, int c) {
          int len;
          int i;

          len = r * c;

          for(i = 0; i < len; i = i + 1) {
            $x = $x + $y;
            x = x.+;
            y = y.+;
          }
        }


        /* NAME: multiplyFloatMatrices
           ARGUMENTS: Pointer to float matrix (@@x), pointer to float matrix (@@y)
                      number of rows (int), number of columns (int)
           EXECUTES: Multiply each element in matrix x with its corresponding element
                      in matrix y. Updates x in memory with the sum.
           RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
           NOTE: Both tuples must have the same number of rows and columns. This is why the this are
                 only passed in once, because they are expected to be the same.
        */
        void multiplyFloatMatrices(float[][] x, float[][] y, int r, int c) {
          int len;
          int i;

          len = r * c;

          for(i = 0; i < len; i = i + 1) {
            $x = $x * $y;
            x = x.+;
            y = y.+;
          }
        }


        /* NAME: divideFloatMatrices
           ARGUMENTS: Pointer to float matrix (@@x), pointer to float matrix (@@y)
                      number of rows (int), number of columns (int)
           EXECUTES: Divide each element in matrix x with its corresponding element
                      in matrix y. Updates x in memory with the sum. Tosses the remainder.
           RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
           NOTE: Both tuples must have the same number of rows and columns. This is why the this are
                 only passed in once, because they are expected to be the same.
        */
        void divideFloatMatrices(float[][] x, float[][] y, int r, int c) {
          int len;
          int i;

          len = r * c;

          for(i = 0; i < len; i = i + 1) {
```

```
          $x = $x / $y;
          x = x.+;
          y = y.+;
        }
      }

      /* NAME: threshHold
         ARGUMENTS: Pointer to tuple matrix (@@@x), number of rows (int), number of columns
 (int),
                    length of tuple(int), pointer to tuple with lowerbound values
 (@lowerBounds),
                    pointer to tuple with upperbound values
         EXECUTES: If the red, green, and blue elements are all within the ranges for the
 tuples
                    (red bound, green bound, blue bound) then all values are set to 0 (ie
 black pixel)
         RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
      */
      void threshHold(int[][][] x, int r, int c, int l, int[] lowerBounds, int[] upperBounds)
 {
        int len;
        int i;
        int j;
        int  k;
        int[3] pix;
        int el1;
        int el2;
        int el3;
        int avg;
        int[][][] a;
        int rLower;
        int gLower;
        int bLower;
        int rUpper;
        int gUpper;
        int bUpper;

        rLower=$lowerBounds;
        lowerBounds=lowerBounds.+;
        gLower=$lowerBounds;
        lowerBounds=lowerBounds.+;
        bLower=$lowerBounds;

        rUpper=$upperBounds;
        upperBounds=upperBounds.+;
        gUpper=$upperBounds;
        upperBounds=upperBounds.+;
        bUpper=$upperBounds;

        len = r * c * l;


        for (i = 0; i < len / 3; i = i + 1) {
          a = x;
          el1 = $a;
          a = a.+;
          el2 = $a;
          a= a.+;
          el3 = $a;
           if (el1>rLower && el1<rUpper && el2>gLower && el2<gUpper && el3>bLower &&
      el3<bUpper){
            x = x.+;
            x = x.+;
            x = x.+;
          }
          else {
            $x=0;
            x = x.+;
            $x=0;
            x = x.+;
            $x=0;
            x = x.+;
```

```
  }

 }

}


/* NAME: subtractFloatMatrices
   ARGUMENTS: Pointer to float matrix (@@x), pointer to float matrix (@@y)
              number of rows (int), number of columns (int)
   EXECUTES: Subtracts each element in matrix x with its corresponding element
              in matrix y. Updates x in memory with the sum.
   RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned
   NOTE: Both tuples must have the same number of rows and columns. This is why the this are
          only passed in once, because they are expected to be the same.
*/
void subtractFloatMatrices(float[][] x, float[][] y, int r, int c) {
 int len;
 int i;

 len = r - c;

 for(i = 0; i < len; i = i + 1) {
  $x = $x / $y;
  x = x.+;
  y = y.+;
 }
}

/* Scalar Operations */

/* NAME: scalarMultiplyIntMatrix
   ARGUMENTS: Pointer to int matrix (@@@x), number of rows(int), number of columns(int), scalar (int).
   EXECUTES: Scalar multiplies an int matrix
```

```
          RETURNS: VOID; the matrix passed in is updated in memory; nothing is returned.
       */
       void scalarMultiplyIntMatrix(int[][] x, int r, int c, int scalar) {
         int i;
         int j;

         for (i = 0; i < r; i = i + 1) {
           for (j = 0; j < c; j = j + 1) {
             $x = $x * scalar;
             x = x.+;
           }
         }
       }

       /* NAME: scalarMultiplyFloatMatrix
          ARGUMENTS: Pointer to float matrix (@@@x), number of rows(int), number of columns(int), scalar (float).
          EXECUTES: Scalar multiplies a float matrix
          RETURNS: VOID; the matrix passed in is updated in memory; nothing is returned.
       */
       void scalarMultiplyFloatMatrix(float[][] x, int r, int c, float scalar) {
         int i;
         int j;

         for (i = 0; i < r; i = i + 1) {
           for (j = 0; j < c; j = j + 1) {
             $x = $x * scalar;
             x = x.+;
           }
         }
       }

       /* NAME: bwIntTupleMatrix
          ARGUMENTS: Pointer to float matrix (@@@x), number of rows(int), number of columns(int), length of tuple (int) number of rows (int), number of columns (int).
          EXECUTES: Makes a matrix "black" and "white" by taking each int tuple, and getting
 the RGB average, and updated each element in the int tuple with calculated average.
          RETURNS: VOID; the first matrix passed in is updated in memory; nothing is returned.
          NOTE: This is intended for an int tuple matrix where the tuple's length=3.
       */

       void bwIntTupleMatrix(int[][][] x, int r, int c, int l) {
         int len;
         int i;
         int j;
         int  k;
         int[3] pix;
         int el1;
         int el2;
         int el3;
         int avg;
         int[][][] a;

         len = r * c * l;

         for (i = 0; i < len / 3; i = i + 1) {
           a = x;
           el1 = $a;
           a = a.+;
           el2 = $a;
           a= a.+;
           el3 = $a;
           avg = (el1+el2+el3)/3;

           $x=avg;
           x = x.+;
           $x =avg;
           x = x.+;
           $x = avg;
           x = x.+;
         }
```

```

      }

      /* NAME: detThree
         ARGUMENTS: Pointer to int matrix (@@x)
         EXECUTES: Calculates the determinant of a 3x3 matrix using Sarrus's rule
         RETURNS: INT; resturns the determinant (an integer)
         NOTE: Only accepts 3x3 matrix because Sarrus's Rule only for 3x3.
      */
      int detThree(int[][] x){
        int a;
        int b;
        int c;
        int d;
        int e;
        int f;
        int g;
        int h;
        int i;
        int sum;
        a = $x;
        x = x.+;
        b = $x;
        x = x.+;
        c = $x;
        x = x.+;
        d = $x;
        x = x.+;
        e = $x;
        x = x.+;
        f = $x;
        x = x.+;
        g = $x;
        x = x.+;
        h = $x;
        x = x.+;
```

```
  i = $x;

  sum = (a * e * i) + (b * f * g) + (c * d * h) - (c * e * g) - (a * f * h) - (b * d * i);

  return sum;
}


/* NAME: out
   ARGUMENTS: Pointer to int tuple matrix (@@@m), int rows, int columns, int length
   EXECUTES: Writes an int tuple matrix out to an image, in the format of a  P3 ppm file.
   RETURNS: VOID
   NOTE: Only accepts int tuple matrix because P3 ppm file is composed of pixels. */
void out(int[][][] m, int r, int c, int l) {
  int i;
  int j;
  int k;

  prints("P3");
  printsl(r);
  printssl(" ");
  print(c);
  prints("255");

  for (i = 0; i < r; i = i + 1) {
   for (j = 0; j < c; j = j + 1) {
     for (k = 0; k < l; k = k + 1) {
       printsl($m);
       printssl(" ");
       m = m.+;
     }
   }
   prints("");
  }
```

```
    }

    /* NAME: printIntTupleMatrix
       ARGUMENTS: Pointer to int tuple matrix (@@@m), int rows, int columns, int length
       EXECUTES: Prints an int tuple matrix
       RETURNS: VOID */
    void printIntTupleMatrix(int[][][] x, int r, int c, int l) {
      int i;
      int j;
      int k;

      printssl("[| ");

      for (i = 0; i < r; i = i + 1) {
        for (j = 0; j < c; j = j + 1) {
          printssl("(");
          for (k = 0; k < l; k = k + 1) {
            if (k != l - 1) {
              printsl($x);
              printssl(", ");
              x = x.+;
            } else {
              printsl($x);
              printssl(")");
            }
          }
          if (j != c - 1) {
            printssl(", ");
            x = x.+;
          } else {
            if (i != r - 1) {
              prints("");
            }
          }
        }
        if (i != r - 1) {
```

```
      printssl(" | ");
    }
    x = x.+;
  }

  prints(" |]");
}

/* NAME: printFloatTupleMatrix
   ARGUMENTS: Pointer to float tuple matrix (@@@m), int rows, int columns, int length
   EXECUTES: Prints a float tuple matrix
   RETURNS: VOID */
void printFloatTupleMatrix(float[][][] x, int r, int c, int l) {
  int i;
  int j;
  int k;

  printssl("[| ");

  for (i = 0; i < r; i = i + 1) {
    for (j = 0; j < c; j = j + 1) {
      printssl("(");
      for (k = 0; k < l; k = k + 1) {
        if (k != l - 1) {
          printfsl($x);
          printssl(", ");
          x = x.+;
        } else {
          printfsl($x);
          printssl(")");
        }
      }
      if (j != c - 1) {
        printssl(", ");
        x = x.+;
      } else {
        if (i != r - 1) {
          prints("");
        }
      }
    }
    if (i != r - 1) {
      printssl(" | ");
    }
    x = x.+;
  }

  prints(" |]");
}

/* NAME: printCharTuple
   ARGUMENTS: Pointer to char tuple (@m), int length
   EXECUTES: Prints a tuple of chars
   RETURNS: VOID */
void printCharTuple(char[] x,  int len) {
  int i;
  char c;

  for (i = 0; i < len; i = i + 1) {
    c = $x;
    printcsl(c);

    x = x.+;
  }
}

/* NAME: printBoolTuple
   ARGUMENTS: Pointer to bool tuple (@m), int length
   EXECUTES: Prints a tuple of bools
   RETURNS: VOID */
void printBoolTuple(bool[] x,  int len) {
  int i;
```

```
     bool b;

     printssl("[");

     for (i = 0; i < len; i = i + 1) {
       b = $x;
       if (b) {
         printssl("true");
       } else {
         printssl("false");
       }

       if (i != len - 1) {
         printssl(", ");
       }

       x = x.+;
     }

     prints("]");
    }

    /*NAME: vertMean
     ARGUMENTS: Pointer to Int Tuple Matrix (@@@variable_name), number of rows (int),
                number of columns (int), length of tuples (int)
     EXECUTES: Iterates over an int tuple matrix and calculates the mean value for
               each integer and replaces the integer in the tuple of focus with this mean value.
               The mean value of each integer is calculated by take the mean value of the int in question,
               the corresponding integer in the row above, and the corresponding integer in the row below.
     RETURNS: VOID
    */

    void vertMean(int[][][] x, int r, int c, int l){
       int i;
       int j;
       int k;
       int t;
       int b;
       int temp;
       int[][][] before;
       int[][][] current;
       int[][][] next;
       int[][][] origin;
       int sum;

       origin = x;
       for (i = 0; i < c-1; i = i +1){
           before = origin;
          for(j = 0; j<(i*l); j = j+1){
               before = before.+;}
           current = before;
           next = before;
           for(k = 0; k < c*l; k = k + 1){
              current =  current.+;
              next = next.+;
              next = next.+;}
           for(t = 0; t<(r-2); t = t + 1){
               for(b = 0; b < l; b = b + 1){
                   sum  = $before + $current + $next;
                   temp = (sum/3);
                   $current = temp;
                   before = before.+;
                   current = current.+;
                   next = next.+;
               }
                for(b = 0; b<(c*(l)-2); b = b+1){
                   before = before.+;
                   current = current.+;
                   next = next.+;}
```

```
          }
    }
}




void rotateIntTupMatrixRight(int[][][][] old, int[][][][] new, int r, int c, int l){

    int[][][][] iter;
    int rowIt;
    int colIt;
    int newCnt;
    int tupCnt;

    for(rowIt=0; rowIt<r; rowIt=rowIt+1){
        for(colIt=0; colIt<c; colIt=colIt+1){
            iter=new;
            for(newCnt=0; newCnt<((((c)-rowIt)+((colIt)*(r)))*l); newCnt=newCnt+1){
                iter = iter.+;
            }
            for(tupCnt=0; tupCnt<l; tupCnt=tupCnt+1){
                $iter=$old;
                iter=iter.+;
                old=old.+;
            }
        }
    }
}
```

4. Project Plan

4.1 Project Overview

The ML development group met 2 to 3 times per week. Our regularly scheduled meeting took place every Monday from 4:00 to 5:00 p.m. Additionally, our group met with the TA who oversaw our group, Jacob Graff, on Wednesdays at 5:20 p.m. During our first meeting as a group, we discussed possible project ideas and talking about a rough timeline for the major milestones of the project. Then, within the first two weeks of the project, we created a Google Calendar that included the dates and times of our weekly meetings, the hard due dates for each milestone of the project (proposal, Language Reference Manual, "Hell World"), as well as goal deadlines for each part of the project and element of our language. Before the first major milestone was due, we brought up many possible ideas for a potential language, finally settling on the inspiration for ML: a matrix language that made image manipulation simple and efficient. The next major milestone, the Language Reference Manual, or LRM, came as somewhat of a challenge, as the group had to really hone in on the defining factors of ML, as well as the features that we could realistically implement in a language with our approximate three-month timeline.

After the LRM was submitted, we got to work on setting up two things: the front end (Scanner, Parser, and AST) and the integration testing. Then, in working towards the "Hello World" milestone, we began the back-end code generation as well. As intended by the nature of the "Hello World" assignment, we were able to build a small version of our future language that went all the way down to producing working LLVM code. After this milestone was reached, we were able to expand out, creating our tuple and matrix type and implementing various operators.

During the last few weeks before the deadline, we worked to perfect all matrix and tuple operations, as well as get file input working for image processing. Additionally, we wrote code in ML to build up our matrix-oriented standard library to implement functions like matrix multiplication, SOMETHING ELSE, and SOMETHING ELSE.

4.2 Project Timeline

- *January 27, 2016*: First meeting with group, got to know each other and decided to move forward as a group.
- *February 1, 2016*: Held a meeting to brainstorm project ideas. Decided on a weekly meeting time (Mondays at 4 p.m. directly after lecture). Established a group GroupMe for messaging, as well as created a google drive folder that the entire group had access to for idea sharing and working on the proposal.
- *February 8, 2016*: Decided to implement a matrix and image, C-like language. Created a rough draft of the proposal.
- *February 9, 2016*: Polished up the proposal and decided on roles for the project.
- *February 10, 2016*: Proposal Due at 2:40 p.m.
- *February 13, 2016*: Established a Wednesday 5:20 p.m. weekly meeting time with Jacob.
- *February 17, 2016*: First meeting with Jacob, discussed feedback of our proposal and direction to go for the next milestone, the LRM.
- *February 22, 2016*: Made decisions regarding the feedback given to us from the instructors on our proposal. Discussed development environment and set up a git repository.
- *Feb 29, 2016:* Began work on the final copy of the LRM.
- *March 2, 2016:* Continued work on the LRM.
- *March 7, 2016:* LRM due.
- *March 9, 2016:* Began work on the scanner, parser and AST.
- *March 21, 2016:* First group meeting after returning from spring break. Discussed progress, impending milestones and re-evaluated timelines.
- *March 23, 2016:* Met with Jacob to discus feedback on our LRM. Afterwards, met as a group to discuss changes we would need to make and how we would implement the instructors' feedback.
- *March 28, 2016:* Met to discuss steps needed to take to finish the "Hello World" demo on time.
- *March 31, 2016:* Met as a group for the evening to work on codegen.ml. Produced "Hello World"
- *April 4, 2016:* Met for our normal weekly meeting to discuss the direction of our project after completing "Hello World". Had two main priorities: tuples and matrices and file input.
- *April 7, 2016:* "Hello World" due.
- *April 11, 2016:* Weekly meeting, discussed progress on main objectives including matrices and tuples, file input, and binops.
- *April 18, 2016:* Tuple and Matrix declaration completed.
- *April 21, 2016:* File input completed.
- *April 29, 2016:* Tuple and matrix access and element assignment completed.
- *April 30, 2016:* Began work on final report.
- *May 2, 2016:* Divided work for standard library functions as well as code examples.

4.3 Challenges and Changes

One of the largest changes we made from Proposal to LRM was our target language. We had originally intended our language to implement parallelism, and as such, our target language was C and we planned to utilize OpenMP for the parallel aspect. However, after receiving feedback about our proposal, we decided it best to change our target language to LLVM. Because of this modification, we changed the syntax of ML from more pythonic to more C-like.

Another major change made along the way was our group decision not to implement parallelism within our language. Because of the change of target language, and the speed at which we progressed throughout the semester, the group decided it would be best not to implement parallelism about three quarters of the way through April. This decision was mostly made so that the group could focus on most important factor of our language: implementing image processing using matrices.

4.4 Roles and Responsibilities

Manager: *Timely completion of deliverables*

- Caroline Trimble

Language Guru: *Language Design*

- Jessica Valarezo

System Architect: *Compiler architecture and development environment*

- Kyle Jackson
- Alex Barkume

Tester: *Test plan, test suites*

- Alex Barkume
- Jared Greene

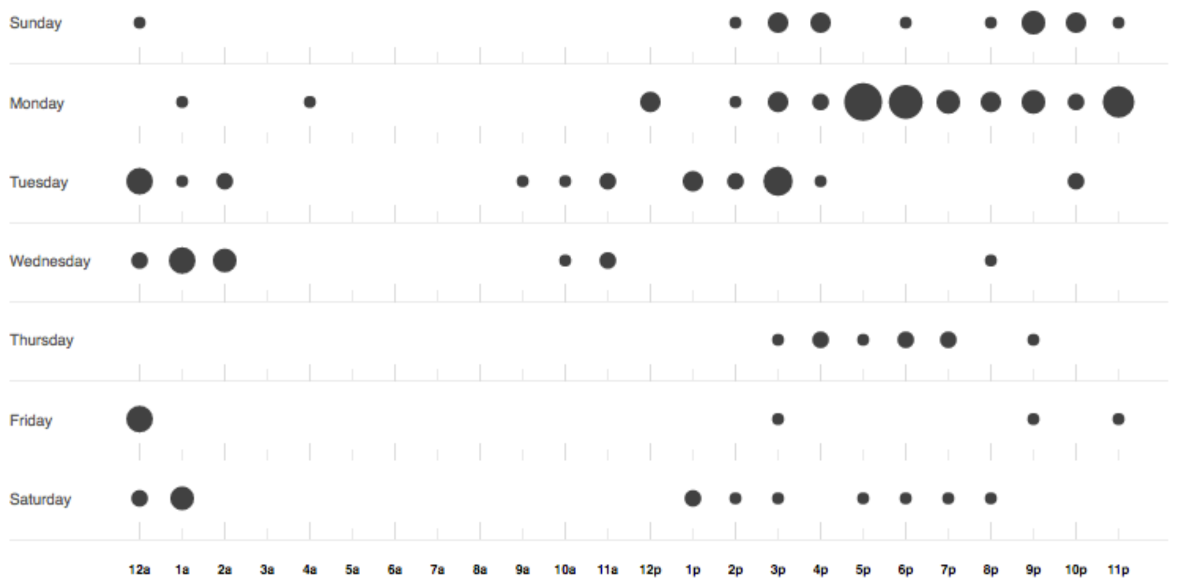
Because we had five group members, while there were only four role descriptions, Alex split up and spent half his time working with Kyle on the compiler architecture, and half his time developing the test plan and test suites.

4.5 Development Environment

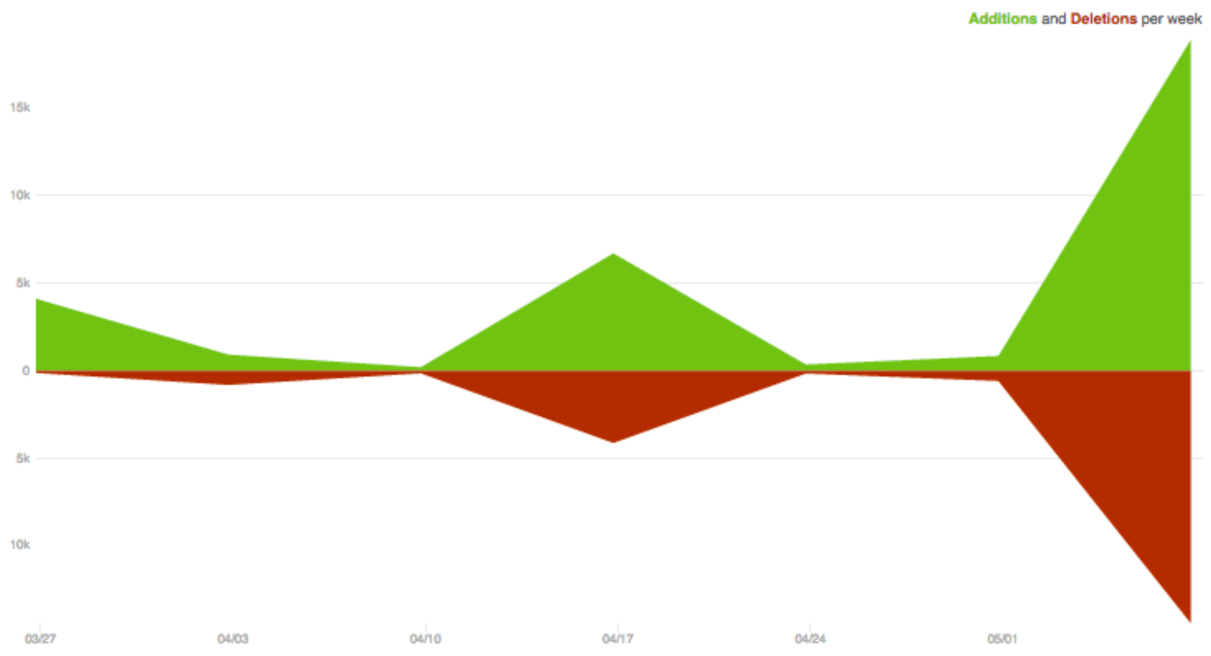
Language	OCaml 4.02.3 with Core 113.00.00
Editor	Sublime, Vim
Version Control	Git
Operating System	OS X 10.11 “El Captain”

4.6 Version Control Graphs and Statistics

4.6.1 Punch Card



4.6.2 Code Frequency



4.6.3 Contributors

4.7 Project Log

- 7cfb619 - Wed May 11 02:11:03 2016 -0400, Caroline Trimble : Merge branch 'master' of https://github.com/kdj2109/ml
- 9387547 - Wed May 11 02:10:27 2016 -0400, kdj2109 : fixed conflict
- 6cc8fe8 - Wed May 11 02:07:16 2016 -0400, kdj2109 : changes
- e5cf31b - Wed May 11 02:04:43 2016 -0400, kdj2109 : fixed prints, added scalar multiplication for int matrices and float matrices to stdlib
- 1c95490 - Wed May 11 01:48:59 2016 -0400, Jared Greene : printIntTupMatrix added
- f3150f4 - Wed May 11 01:20:54 2016 -0400, Jared Greene : Merge branch 'master' of https://github.com/kdj2109/ml
- ea86a56 - Wed May 11 01:20:42 2016 -0400, Jared Greene : merged stdlib
- 2e412c5 - Wed May 11 01:15:15 2016 -0400, kdj2109 : deleted build
- 74125db - Wed May 11 01:14:41 2016 -0400, kdj2109 : fixed some tests
- c1c7259 - Wed May 11 01:13:53 2016 -0400, Jared Greene : printIntTupleMatrix stdlib func and test
- b634a51 - Wed May 11 00:37:57 2016 -0400, Caroline Trimble : Merge branch 'master' of https://github.com/kdj2109/ml
- 8020043 - Wed May 11 00:10:57 2016 -0400, kdj2109 : got rid of object files
- cf7a2aa - Tue May 10 22:50:24 2016 -0400, Jessica Valarezo : Merge branch 'master' of github.com:kdj2109/ml
- d41faf8 - Tue May 10 22:37:04 2016 -0400, Jessica Valarezo : Out function.
- fe14aa6 - Tue May 10 22:24:45 2016 -0400, Jared Greene : printIntMatrices and matrix on matrix functions added to the stdlib
- f25ed19 - Tue May 10 22:18:34 2016 -0400, Jared Greene : tests for int matrix mult & printIntMatrix
- 5c92cf2 - Tue May 10 21:51:51 2016 -0400, Jared Greene : multiply test failing
- 258016f - Tue May 10 21:12:24 2016 -0400, Caroline Trimble : bigger picture
- 211d4d3 - Tue May 10 15:47:38 2016 -0400, Jessica Valarezo : Merge branch 'master' of github.com:kdj2109/ml

- f2f41a0 - Tue May 10 15:47:29 2016 -0400, Jessica Valarezo : Files for testing.
- eb86b97 - Tue May 10 15:46:35 2016 -0400, Jessica Valarezo : Tests for opening an image, file, and converting into black and white.
- b49feb8 - Tue May 10 15:41:50 2016 -0400, Jessica Valarezo : Added in function to turn int tuple matrix to bw.
- 366bf90 - Tue May 10 15:39:39 2016 -0400, Jessica Valarezo : Updated regex for identifying id.
- fca1634 - Tue May 10 14:49:25 2016 -0400, kdj2109 : malloc and free
- c51a382 - Tue May 10 14:12:48 2016 -0400, Jared Greene : Added printIntMatrices to stdlib
- 5d3147d - Tue May 10 13:14:04 2016 -0400, kdj2109 : fixed codegen error for type_of_identifier
- d22b154 - Tue May 10 13:04:53 2016 -0400, kdj2109 : tuple length test
- 23f850a - Tue May 10 11:38:47 2016 -0400, kdj2109 : cleanup semant
- 4884fe1 - Tue May 10 11:17:53 2016 -0400, kdj2109 : fail tests / semant to check tuples / tweaking codegen for valid pointer types and
- 28ad4ea - Tue May 10 10:32:12 2016 -0400, Caroline Trimble : edwards pic
- e1ce9a8 - Tue May 10 09:41:43 2016 -0400, Caroline Trimble : function where assign pointer to new variable
- b1d1e41 - Tue May 10 01:04:00 2016 -0400, Jared Greene : Merge branch 'master' of https://github.com/kdj2109/ml
- f3616d8 - Tue May 10 00:51:41 2016 -0400, kdj2109 : more work
- bffece4 - Tue May 10 00:43:57 2016 -0400, kdj2109 : semant work
- 865ef6d - Tue May 10 00:26:28 2016 -0400, kdj2109 : deleted file
- 841965d - Tue May 10 00:15:42 2016 -0400, kdj2109 : Merge branch 'master' of http://github.com/kdj2109/ml
- b7e7fdc - Tue May 10 00:15:35 2016 -0400, kdj2109 : char/bool tuple test and printc and printcsl functions for printing chars
- 01bd3d0 - Tue May 10 00:03:58 2016 -0400, Jared Greene : Merge branch 'master' of https://github.com/kdj2109/ml Stdlib updated. All arithmetic operations for matrix on matrix and tuple of tuple operations added to stdlib for all valid types of each.
- 4b08e3c - Tue May 10 00:03:52 2016 -0400, Jared Greene : All arithmetic operations for matrix on matrix and tuples on tuples implemented for all types of each

- 7b3e785 - Mon May 9 23:30:26 2016 -0400, Jessica Valarezo : test for open removed for now.
- d6f2e65 - Mon May 9 23:22:43 2016 -0400, Jessica Valarezo : Merge branch 'master' of github.com:kdj2109/ml
- 6e168c6 - Mon May 9 23:22:38 2016 -0400, Jessica Valarezo : Merged.
- 98190cd - Mon May 9 23:21:17 2016 -0400, Caroline Trimble : deleted print filename
- df0648c - Mon May 9 23:20:21 2016 -0400, Jessica Valarezo : Declared a function to convert an int tuple matrix into black and white.
- 38d2f9e - Mon May 9 23:20:07 2016 -0400, Jessica Valarezo : Test for opening a ppm.
- d9a7f59 - Mon May 9 23:19:48 2016 -0400, Jessica Valarezo : Test for converting a matrix into black and white.
- bcd32a0 - Mon May 9 23:18:25 2016 -0400, kdj2109 : fixed test tuple lib
- 77b0ae2 - Mon May 9 23:06:35 2016 -0400, kdj2109 : ...
- 69a0762 - Mon May 9 23:04:33 2016 -0400, kdj2109 : changed print tuple functions in stdlib, added single print statements in semant and codegen, more cleanup
- 45c7522 - Mon May 9 22:52:16 2016 -0400, Caroline Trimble : Merge branch 'master' of https://github.com/kdj2109/ml
- ee649d7 - Mon May 9 22:51:47 2016 -0400, Caroline Trimble : std lib test with global variable
- 063f37c - Mon May 9 22:17:18 2016 -0400, kdj2109 : cleaning up some more
- d15e3e0 - Mon May 9 21:54:24 2016 -0400, kdj2109 : more cleanup
- c232381 - Mon May 9 21:34:27 2016 -0400, kdj2109 : deleted build files
- b544bd2 - Mon May 9 21:33:30 2016 -0400, kdj2109 : more semant
- cc50151 - Mon May 9 20:59:53 2016 -0400, kdj2109 : changed some semantics
- 81c56e6 - Mon May 9 20:05:56 2016 -0400, kdj2109 : changed testall.sh to create and run executables
- 068bb7f - Mon May 9 19:25:10 2016 -0400, kdj2109 : rows, columns for matrices
- a3fab31 - Mon May 9 19:10:08 2016 -0400, Caroline Trimble : picture tests
- feade98 - Mon May 9 18:49:50 2016 -0400, kdj2109 : Merge branch 'master' of http://github.com/kdj2109/ml
- 7e7f393 - Mon May 9 18:49:42 2016 -0400, kdj2109 : semantic checking for pointers/matrix & tuple access

- 994ce45 - Mon May 9 18:33:22 2016 -0400, Caroline Trimble : added more functions to std lib
- d7c0d39 - Mon May 9 18:24:34 2016 -0400, Caroline Trimble : tests matrix add float function
- c9167b6 - Mon May 9 18:23:57 2016 -0400, Caroline Trimble : determinant in function form
- c8c23ed - Mon May 9 18:11:06 2016 -0400, kdj2109 : fixed all the warnings
- 7659662 - Mon May 9 17:24:21 2016 -0400, kdj2109 : changed pointer types to - tuple pointer type, matrix pointer type, and matrix tuple pointer type, each is pointing to the first element in the collection
- 5f089b1 - Mon May 9 17:02:25 2016 -0400, kdj2109 : matrix functions done
- 6e59c7b - Mon May 9 15:06:52 2016 -0400, Caroline Trimble : lol whoops ended a comment wrong
- c2d6298 - Mon May 9 14:57:05 2016 -0400, kdj2109 : Merge branch 'master' of http://github.com/kdj2109/ml
- e3514d9 - Mon May 9 14:56:39 2016 -0400, kdj2109 : add int matrices
- 9244b1b - Mon May 9 12:56:10 2016 -0400, Caroline Trimble : standard library with comment formatting
- 8a34433 - Mon May 9 12:47:20 2016 -0400, Caroline Trimble : test that utilizes the std lib for tuples
- 911b701 - Mon May 9 12:46:36 2016 -0400, Caroline Trimble : standard lib w prints and add tup
- 106e9e6 - Sun May 8 23:13:00 2016 -0400, Caroline Trimble : flower picture
- 71de66c - Sun May 8 22:37:57 2016 -0400, Caroline Trimble : print fxn for integer tuples
- 5579ec2 - Sun May 8 22:28:29 2016 -0400, Caroline Trimble : tests for print float method for tuples...unfortunately all on new lines
- 2e6ba5c - Sun May 8 22:26:46 2016 -0400, Caroline Trimble : determinant test and access/assign matrices with tuples
- cdee1ca - Sun May 8 21:33:25 2016 -0400, kdj2109 : pointer arithmetic for tuples works
- bceaa84 - Sun May 8 16:06:07 2016 -0400, kdj2109 : deleted ll file
- 1a13094 - Sun May 8 16:05:06 2016 -0400, kdj2109 : progress with pointers
- a7f6aff - Sun May 8 15:25:51 2016 -0400, kdj2109 : dereferencing a ptr
- 18c7bca - Sun May 8 00:07:41 2016 -0400, ajb2233 : Non-functional matrix return type (tried using pointer to a pointer)

- 158a95b - Sat May 7 20:19:36 2016 -0400, kdj2109 : returning pointer works
- f18628b - Sat May 7 19:46:24 2016 -0400, kdj2109 : ... working on functions ...
- ef301f9 - Sat May 7 17:43:51 2016 -0400, kdj2109 : can pass in ptrs for tuple functions -- use @sign to ' reference ' the tuple
- 065969e - Sat May 7 15:49:42 2016 -0400, kdj2109 : finished length
- f957fcb - Sat May 7 13:52:50 2016 -0400, kdj2109 : front end - length
- 9a73530 - Sat May 7 01:50:53 2016 -0400, kdj2109 : Merge branch 'master' of http://github.com/kdj2109/ml
- 7f016ad - Sat May 7 01:50:40 2016 -0400, kdj2109 : functions with tuples beginning to work
- 99f1ab4 - Sat May 7 01:27:55 2016 -0400, Jessica Valarezo : Testing files for io and includes
- 418b442 - Sat May 7 01:21:19 2016 -0400, Jessica Valarezo : Proper formatting according to new matrix declaration.
- 66d6f8c - Sat May 7 01:20:36 2016 -0400, Jessica Valarezo : Updated variable names and formatting.
- ba81759 - Fri May 6 21:40:30 2016 -0400, Caroline Trimble : editng open
- 5069b9b - Fri May 6 20:20:35 2016 -0400, Jessica Valarezo : Merge branch 'I/O'
- 51d9941 - Fri May 6 20:17:51 2016 -0400, Jessica Valarezo : Merge branch 'master' of github.com:kdj2109/ml
- 515521b - Fri May 6 00:54:16 2016 -0400, Jessica Valarezo : Testing script had to be updated to use sys.argv isntead of piping.
- 74f70c9 - Fri May 6 00:53:37 2016 -0400, Jessica Valarezo : File modified to read file from sys.argv.(1) instead of from stdin.
- 616ce76 - Fri May 6 00:53:00 2016 -0400, Jessica Valarezo : Makefile links ml.ml file to prep.ml module.
- c4ed698 - Fri May 6 00:52:31 2016 -0400, Jessica Valarezo : Include and Io were combined into prep file.
- a91dd20 - Fri May 6 00:51:54 2016 -0400, Jessica Valarezo : Test files for includes and open updated.
- 01e8bcf - Wed May 4 11:21:46 2016 -0400, Caroline Trimble : #include with new lines
- 30360e6 - Wed May 4 11:13:24 2016 -0400, Caroline Trimble : includes statement without newlines
- 593bddb - Wed May 4 10:42:57 2016 -0400, Jessica Valarezo : Reversed the list of pixels to match .ppm file.

- a5c51f1 - Sat Apr 30 18:32:56 2016 -0400, ajb2233 : Complete implementation of matrices for types: int, float, int tuple, float tuple. Note: Matrix of tuples notation could probably be improved.
- 067244a - Sat Apr 30 13:56:53 2016 -0400, Jessica Valarezo : Opens file specified in ppm, creates matrix declaration and writes to file/temp.mxl file.
- 424bded - Sat Apr 30 00:48:18 2016 -0400, ajb2233 : Float matrix declaration now functional. Accessing not yet functional.
- a443770 - Sat Apr 30 00:12:38 2016 -0400, ajb2233 : Now assignment works too
- 597f565 - Fri Apr 29 23:59:58 2016 -0400, ajb2233 : Implentation of mxn integer matrices (for real this time) and working access
- ba8d549 - Thu Apr 28 15:55:17 2016 -0400, Jessica Valarezo : Reads in ppm and creates matrix declaration string.
- a9d00d3 - Wed Apr 27 02:00:43 2016 -0400, kdj2109 : f
- b8ec339 - Wed Apr 27 00:31:02 2016 -0400, Jessica Valarezo : Reads in ppm and creates a string of digits.
- 5479997 - Mon Apr 25 18:10:08 2016 -0400, Jessica Valarezo : Merge branch 'I/O' of github.com:kdj2109/ml into I/O
- 91c5632 - Mon Apr 25 18:10:02 2016 -0400, Jessica Valarezo : pixel list.
- 1441453 - Mon Apr 25 18:04:17 2016 -0400, Caroline Trimble : new regexp
- a8222e8 - Mon Apr 25 17:56:47 2016 -0400, Jessica Valarezo : Opens ppm file
- c241c33 - Mon Apr 25 17:43:44 2016 -0400, Caroline Trimble : RIGHT pic
- b465854 - Mon Apr 25 17:41:36 2016 -0400, Caroline Trimble : pic
- aec122f - Mon Apr 25 17:40:39 2016 -0400, Caroline Trimble : Merge branch 'I/O' of https://github.com/kdj2109/ml into I/O
- be4a211 - Mon Apr 25 17:39:39 2016 -0400, Caroline Trimble : picture
- 7235b48 - Mon Apr 25 17:02:10 2016 -0400, Jessica Valarezo : Str module works.
- fafa9d9 - Sun Apr 24 21:24:07 2016 -0400, Jessica Valarezo : I/O compiles
- 333c400 - Sun Apr 24 20:31:16 2016 -0400, Caroline Trimble : io files
- 73b6cc2 - Sun Apr 24 18:58:18 2016 -0400, Jessica Valarezo : Updated with Jared's code.
- 49caccc - Sun Apr 24 17:19:37 2016 -0400, Jared Greene : Merge branch 'I/O' of https://github.com/kdj2109/ml into I/O

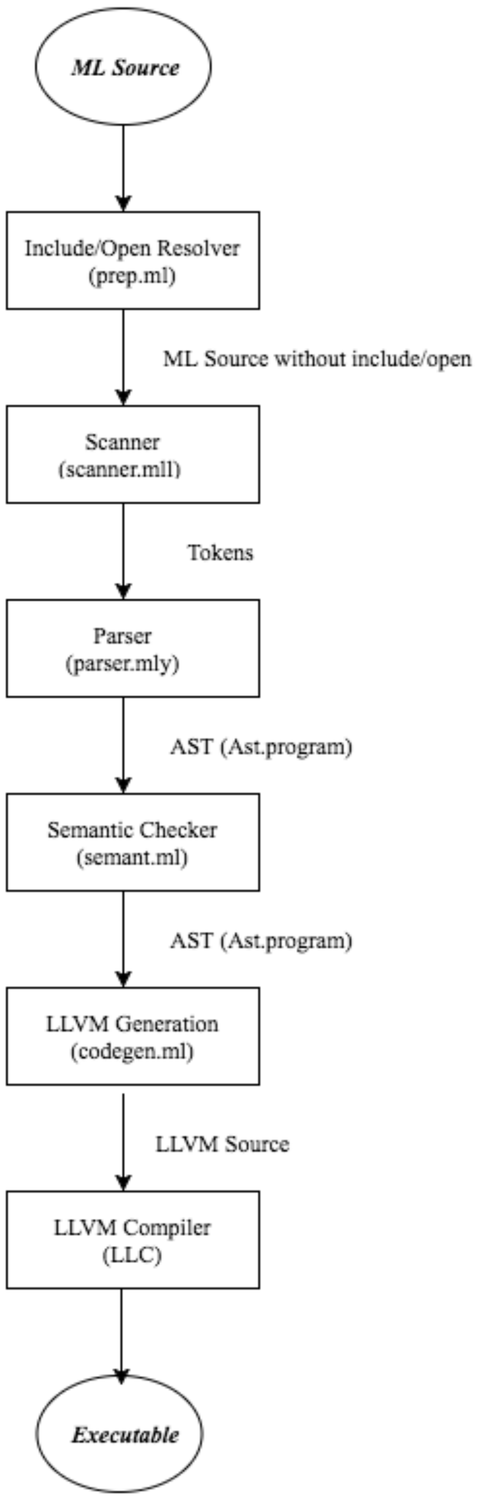
- c9574da - Sun Apr 24 17:18:14 2016 -0400, Jared Greene : open added to semnt as built in
- a0ac117 - Sun Apr 24 16:22:09 2016 -0400, Jessica Valarezo : Merge branch 'I/O' of github.com:kdj2109/ml
- 223331a - Sun Apr 24 15:55:30 2016 -0400, Jared Greene : open added to semnt as built in
- 9858bc3 - Wed Apr 20 20:53:28 2016 -0400, kdj2109 : changed matrix to be [|...|r, c] -- if you want to declare it with tuples do [|(...), (...), ...|]r, c] while tuples are just [...]
- f2a82be - Wed Apr 20 01:18:14 2016 -0400, kdj2109 : changed matrix primitives to [|...|dim_tuple, rows, columns|] -> free to change however but makes it easier to determine when it is a matrix of primitives or matrix of tuples as we are storing everything linearly -- semant checking for tuples and matrices is pretty much done
- e78fe54 - Tue Apr 19 16:57:22 2016 -0400, kdj2109 : changes everything to linear :)
- fd0504e - Tue Apr 19 15:35:17 2016 -0400, kdj2109 : added matrix frontend, started adding matrix backend
- 26ec11c - Tue Apr 19 01:32:16 2016 -0400, kdj2109 : added matrix type, matrix access, using linear storage for matrices, started tuple access -- giving me some errors
- 1e9d170 - Mon Apr 18 19:55:52 2016 -0400, kdj2109 : deleted build files
- a30d65a - Mon Apr 18 19:54:04 2016 -0400, kdj2109 : refactor of front and backend, tuples dec and init
- d8f91ee - Mon Apr 18 15:09:40 2016 -0400, ajb2233 : Declaration and initialization of mxn matrices implemented. Mxn matrix literal in following format: {1,2,..,n | 1,2,..n | ... | m}
- 34f57a6 - Sun Apr 17 14:57:09 2016 -0400, ajb2233 : Declaration and initilization of 1 by n integer matrices implemented.
- 8898a76 - Thu Apr 14 21:37:45 2016 -0400, Jessica Valarezo : Merge branch 'master' of github.com:kdj2109/ml
- 661097c - Thu Apr 14 21:34:27 2016 -0400, Jessica Valarezo : Added matching for tuple.
- cc367b4 - Tue Apr 12 13:15:33 2016 -0400, kdj2109 : tuple declaration changed to primivitive<length> ID -- tuple type added to codegen -- shift/reduce & reduce/reduce conflicts resolved -- tuple declaration tested -- still need tuple assignment, etc.
- 989d431 - Mon Apr 11 17:37:33 2016 -0400, Jessica Valarezo : Tupleeee.

- 14e6bd0 - Mon Apr 11 17:14:20 2016 -0400, Jessica Valarezo : Includes datatype instead of typ.
- fe974f0 - Mon Apr 11 17:08:12 2016 -0400, Jessica Valarezo : Updated codegen for literals.
- 8717bf9 - Mon Apr 11 17:04:31 2016 -0400, Jessica Valarezo : Updated parser for tuple type.
- 39a1b98 - Mon Apr 11 16:59:42 2016 -0400, Jessica Valarezo : Added in tuple primitive.
- f5ea749 - Mon Apr 4 18:40:05 2016 -0400, kdj2109 : changed program filename ext from .mc to .mxl
- 68d8452 - Mon Apr 4 16:29:12 2016 -0400, kdj2109 : changing microc to ml
- 1ca34e5 - Mon Apr 4 15:13:58 2016 -0400, kdj2109 : unops for ints and floats
- ade0ec9 - Sun Apr 3 16:35:34 2016 -0400, kdj2109 : binary operations for floats
- 778a458 - Sat Apr 2 14:04:14 2016 -0400, ajb2233 : PrettyPrint statements completed, only thing remaining is expressions, which will take some tinkering around of the existing prettyprint code.
- be30a58 - Fri Apr 1 15:49:50 2016 -0400, ajb2233 : Start of prettyprint tree for AST implemented. Currently just formats program so has correct tabs. Next step I will implement shortly is breaking up things that are currently represented as one node down further (eg if(x==5) is currently one node, will make that so it's 2 or 3).
- 9bb2fb7 - Thu Mar 31 19:07:30 2016 -0400, kdj2109 : added arith operators for float in semant.ml
- 8cde3bd - Thu Mar 31 19:00:54 2016 -0400, kdj2109 : changed hello world tests
- 7a2b88f - Thu Mar 31 18:56:58 2016 -0400, kdj2109 : Merge branch 'master' of http://github.com/kdj2109/ml merging
- 746225f - Thu Mar 31 18:56:33 2016 -0400, kdj2109 : added string type in parser
- 61c6051 - Thu Mar 31 18:47:04 2016 -0400, Caroline Trimble : no quotes on helloworld
- 5a2583f - Thu Mar 31 17:08:09 2016 -0400, kdj2109 : llvm resources in readme
- 13a0384 - Thu Mar 31 16:49:22 2016 -0400, kdj2109 : deleted func
- 11ac61d - Thu Mar 31 16:47:22 2016 -0400, kdj2109 : scanner cleanup
- 69dc2d1 - Wed Mar 30 02:08:06 2016 -0400, kdj2109 : hello world working -> has quotes around the string, need to fix
- 3069c1f - Tue Mar 29 15:59:22 2016 -0400, kdj2109 : pretty printing ast

- edf606a - Tue Mar 29 02:55:23 2016 -0400, kdj2109 : pass rate status
- d0c2b6b - Tue Mar 29 02:20:10 2016 -0400, kdj2109 : printf for floats now works --> float arithmetic does not
- e2b973f - Mon Mar 28 21:46:29 2016 -0400, kdj2109 : string additions
- 52be1ad - Mon Mar 28 20:36:49 2016 -0400, kdj2109 : commit
- 9c09bd6 - Mon Mar 28 04:24:25 2016 -0400, kdj2109 : began adding float type
- 84a298c - Mon Mar 28 01:16:40 2016 -0400, kdj2109 : added hello world test
- 968dd99 - Sun Mar 27 21:48:41 2016 -0400, kdj2109 : added ml symbols to parser and scanner -- all tests pass
- b92028d - Sun Mar 27 21:14:40 2016 -0400, kdj2109 : initial commit

5. Architecture

5.1 Architecture Diagram



5.2 Toplevel – ml.ml

The Ml module contains the entry point to our compiler. First, it passes the file name as a command line argument to prep.ml, which resolves the include and open statements, which produces a string of the program. It then calls each of the subsequent steps of of the compiler. If any stage encounters an unrecoverable error, it throws an exception. The toplevel produces LLVM code, when then gets compiled using the LLC compiler and produces an executable.

5.3 Include/Open Resolver – prep.ml

The Include/Open Resolver first, looks through the input .mxl file for any #include statements. Upon finding an include statement, it opens that file and pastes the files contents into the input .mxl file. If the file that was included has any include statements itself, those too are included, and so on. Then, the program scans the file for any open(filename.ppm) statements. Upon finding an open statement, the program opens the .ppm file and creates a matrix declaration based on its contents.

5.4 Scanner – scanner.mll

The scanner's task is to generate tokens before being passed to the parser. Tokens include: keywords, identifiers, operators, literals, and symbols. The scanner also converts escape sequences in string literals, removes comments, and removes whitespace (spaces, tabs, newlines). After it completes these tasks, it passes the newly generated tokens to the parser.

5.5 Parser – parser.mly

The parser takes as input the tokens from the scanner, it then creates the abstract syntax tree (AST) based on these tokens.

5.6 Semantic Checker – semant.ml

The semantic checker's role is to enforce the rules for Matrices Language that were explained in the Language Reference Manual, included at the beginning of this document. If the AST complies with these rules, semant.ml will return void. Otherwise, semant.ml will raise an exception. This is the last place where a compiler-time will find an error. If the AST is passed from semant.ml to codegen.ml, LLVM will be generated.

5.7 LLVM Generation – codegen.ml

The code generator uses the semantic abstract syntax tree passed to it by semant.ml to construct the LLVM IR, which is the final stage of the compiler. The LLVM generator iterates through the abstract syntax tree and produces the equivalent LLVM code for each function, statement, or expression. Once the inheritance code is generated, the code generator iterates through the entire semantic abstract syntax tree and produces the necessary LLVM code for each function, statement, and expression. This is done using the OCaml LLVM module. The LLVM code produced from codegen.ml can then be compiled using the LLVM compiler, LLC, to produce an executable which the user can run.

6. Test Plan

6.1 Test Example 1

Test Name: test-tuplemat2
Description: This is a test that declares a matrix of float tuples and then accesses and reassigns elements of that matrix and prints the results out.

6.1.1 *Source Code in Native Language*

```
int main() {
    float[3][2:2] x;
    float b;
    float[3] o;
    float o1;
    float o2;
    float[3] c;
    float[3] i;
    x=[|(11.11,22.22,33.33),(55.55,66.66,77.77)|(17.17,38.38,17.18),(17.37,38.55,35.35)|];
        x[0:0]= [45.5, 54.2, 33.33];
        c = x[0:0];
        b= c[0];
        printf(b);
    b = c[1];
    printf(b);
    o = x[1:0];
    o1 = o[0];
    o2 = o[1];
    o[0] = o1;
    o[1] = o2;
    o[2] = 22.3;
    x[1:0] = o;
    c = x[1:0];
    b = c[2];

    printf(b);
    return 0;
}
```

6.1.2 *Source Code in Target Language*

```
; ModuleID = 'ML'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%f\0A\00"

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

define i32 @main() {
entry:
  %x = alloca [2 x [2 x [3 x double]]]
  %b = alloca double
  %o = alloca [3 x double]
  %o1 = alloca double
  %o2 = alloca double
  %c = alloca [3 x double]
  %i = alloca [3 x double]
  store [2 x [2 x [3 x double]]] [[2 x [3 x double]] [[3 x double] [double 1.111000e+01, double 2.222000e+01, double3.333000e+01], [3 x double] [double 5.555000e+01, double 6.666000e+01, double 7.777000e+01]], [2 x [3 x double]] [[3 x double][double 1.717000e+01, double 3.838000e+01, double 1.718000e+01], [3 x double] [double 1.737000e+01, double 3.855000e+01, double3.535000e+01]]], [2 x [2 x [3 x double]]]* %x
  %x1 = getelementptr [2 x [2 x [3 x double]]]* %x, i32 0, i32 0, i32 0
  store [3 x double] [double 4.550000e+01, double 5.420000e+01, double 3.333000e+01], [3 x double]* %x1
  %x2 = getelementptr [2 x [2 x [3 x double]]]* %x, i32 0, i32 0, i32 0
  %x3 = load [3 x double]* %x2
  store [3 x double] %x3, [3 x double]* %c
  %c4 = getelementptr [3 x double]* %c, i32 0, i32 0
  %c5 = load double* %c4
  store double %c5, double* %b
  %b6 = load double* %b
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt1, i32 0, i32 0), double %b6)
```

```
%c7 = getelementptr [3 x double]* %c, i32 0, i32 1
%c8 = load double* %c7
store double %c8, double* %b
%b9 = load double* %b
%printf10 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt1, i32 0, i32 0), double %b9)
%x11 = getelementptr [2 x [2 x [3 x double]]]* %x, i32 0, i32 1, i32 0
%x12 = load [3 x double]* %x11
store [3 x double] %x12, [3 x double]* %o
%o13 = getelementptr [3 x double]* %o, i32 0, i32 0
%o14 = load double* %o13
store double %o14, double* %o1
%o15 = getelementptr [3 x double]* %o, i32 0, i32 1
%o16 = load double* %o15
store double %o16, double* %o2
%o17 = getelementptr [3 x double]* %o, i32 0, i32 0
%o118 = load double* %o1
store double %o118, double* %o17
%o19 = getelementptr [3 x double]* %o, i32 0, i32 1
%o220 = load double* %o2
store double %o220, double* %o19
%o21 = getelementptr [3 x double]* %o, i32 0, i32 2
store double 2.230000e+01, double* %o21
%x22 = getelementptr [2 x [2 x [3 x double]]]* %x, i32 0, i32 1, i32 0
%o23 = load [3 x double]* %o
store [3 x double] %o23, [3 x double]* %x22
%x24 = getelementptr [2 x [2 x [3 x double]]]* %x, i32 0, i32 1, i32 0
%x25 = load [3 x double]* %x24
store [3 x double] %x25, [3 x double]* %c
%c26 = getelementptr [3 x double]* %c, i32 0, i32 2
%c27 = load double* %c26
store double %c27, double* %b
%b28 = load double* %b
%printf29 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt1, i32 0, i32 0), double %b28)
ret i32 0
```

```
}
```

6.2 Test Example 2

Test Name: test-matrixbw
Description: This is a test that declares a matrix of int tuples and then converts them to black or white (useful for converting an image, in the form of an int tuple matrix, to greyscale). It accesses every element, reassigns it with the calculated value, and prints the results out.

6.1.1 *Source Code in Native Language*

```
void bwIntTupMatrix(int[][][] x, int r, int c, int l) {
  int len;
  int i;
  int[3] pix;
  int el1;
  int el2;
  int el3;
  int avg;
  int[][][] a;

  len = r * c * l;

  for (i = 0; i < len / 3; i = i + 1) {
    a = x;
    el1 = $a;
    a = a.+;
    el2 = $a;
    a= a.+;
    el3 = $a;
    avg = (el1+el2+el3)/3;

    $x=avg;
    x = x.+;
    $x =avg;
    x = x.+;
    $x = avg;
    x = x.+;
  }

}

int main() {
  int[3][2:2] a;
  int[3] c;
  int i;
  int j;
  int k;

  a = [| (1, 2, 3), (4, 5, 6) | (1, 2, 3), (4, 5, 6) |];

  bwIntTupMatrix(@@@a, a.rows, a.columns, a.length);
  for (i = 0; i < a.rows ; i = i + 1) {
    for (j = 0; j < a.columns; j = j + 1) {
      for (k = 0; k < a.length; k = k + 1) {
        c=a[i:j];
        print(c[k]);
      }
    }
  }
  return 0;
}
```

6.1.2 *Source Code in Target Language*

```
; ModuleID = 'ML'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt2 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt3 = private unnamed_addr constant [3 x i8] c"%f\00"
@fmt4 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt5 = private unnamed_addr constant [3 x i8] c"%c\00"
```

```
@fmt6 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt7 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt8 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt9 = private unnamed_addr constant [3 x i8] c"%f\00"
@fmt10 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt11 = private unnamed_addr constant [3 x i8] c"%c\00"

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

define i32 @main() {
entry:
  %a = alloca [2 x [2 x [3 x i32]]]
  %c = alloca [3 x i32]
  %i = alloca i32
  %j = alloca i32
  %k = alloca i32
  store [2 x [2 x [3 x i32]]] [[2 x [3 x i32]] [[3 x i32] [i32 1, i32 2, i32 3], [3 x i32] [i32 4, i32 5, i32 6]], [2 x [3 x i32]][[3 x i32] [i32 1, i32 2, i32 3], [3 x i32] [i32 4, i32 5, i32 6]]], [2 x [2 x [3 x i32]]]* %a
  %a1 = getelementptr inbounds [2 x [2 x [3 x i32]]]* %a, i32 0, i32 0, i32 0, i32 0
  call void @bwIntTupMatrix(i32* %a1, i32 2, i32 2, i32 3)
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge20, %entry
  %i23 = load i32* %i
  %tmp24 = icmp slt i32 %i23, 2
  br i1 %tmp24, label %while_body, label %merge25

while_body:                                       ; preds = %while
  store i32 0, i32* %j
  br label %while2

while2:                                           ; preds = %merge, %while_body
  %j18 = load i32* %j
  %tmp19 = icmp slt i32 %j18, 2
  br i1 %tmp19, label %while_body3, label %merge20

while_body3:                                      ; preds = %while2
  store i32 0, i32* %k
  br label %while4

while4:                                           ; preds = %while_body5, %while_body3
  %k14 = load i32* %k
  %tmp15 = icmp slt i32 %k14, 3
  br i1 %tmp15, label %while_body5, label %merge

while_body5:                                      ; preds = %while4
  %i6 = load i32* %i
  %j7 = load i32* %j
  %a8 = getelementptr [2 x [2 x [3 x i32]]]* %a, i32 0, i32 %i6, i32 %j7
  %a9 = load [3 x i32]* %a8
  store [3 x i32] %a9, [3 x i32]* %c
  %k10 = load i32* %k
  %c11 = getelementptr [3 x i32]* %c, i32 0, i32 %k10
  %c12 = load i32* %c11
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt, i32 0, i32
 0), i32 %c12)
  %k13 = load i32* %k
  %tmp = add i32 %k13, 1
  store i32 %tmp, i32* %k
  br label %while4

merge:                                            ; preds = %while4
  %j16 = load i32* %j
  %tmp17 = add i32 %j16, 1
  store i32 %tmp17, i32* %j
  br label %while2

merge20:                                          ; preds = %while2
  %i21 = load i32* %i
  %tmp22 = add i32 %i21, 1
  store i32 %tmp22, i32* %i
  br label %while

merge25:                                          ; preds = %while
  ret i32 0
}

define void @bwIntTupMatrix(i32* %x, i32 %r, i32 %c, i32 %l) {
entry:
  %x1 = alloca i32*
  store i32* %x, i32** %x1
  %r2 = alloca i32
  store i32 %r, i32* %r2
  %c3 = alloca i32
  store i32 %c, i32* %c3
  %l4 = alloca i32
  store i32 %l, i32* %l4
  %len = alloca i32
  %i = alloca i32
  %pix = alloca [3 x i32]
  %el1 = alloca i32
  %el2 = alloca i32
  %el3 = alloca i32
  %avg = alloca i32
  %a = alloca i32*
  %r5 = load i32* %r2
  %c6 = load i32* %c3
  %tmp = mul i32 %r5, %c6
  %l7 = load i32* %l4
  %tmp8 = mul i32 %tmp, %l7
  store i32 %tmp8, i32* %len
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i45 = load i32* %i
  %len46 = load i32* %len
  %tmp47 = sdiv i32 %len46, 3
  %tmp48 = icmp slt i32 %i45, %tmp47
  br i1 %tmp48, label %while_body, label %merge

while_body:                                       ; preds = %while
  %x9 = load i32** %x1
  store i32* %x9, i32** %a
  %a10 = load i32** %a
  %a11 = load i32* %a10
  store i32 %a11, i32* %el1
  %a12 = getelementptr inbounds i32** %a, i32 0
  %a13 = load i32** %a12
  %a14 = getelementptr inbounds i32* %a13, i32 1
  store i32* %a14, i32** %a
  %a15 = load i32** %a
  %a16 = load i32* %a15
  store i32 %a16, i32* %el2
  %a17 = getelementptr inbounds i32** %a, i32 0
  %a18 = load i32** %a17
  %a19 = getelementptr inbounds i32* %a18, i32 1
  store i32* %a19, i32** %a
  %a20 = load i32** %a
  %a21 = load i32* %a20
  store i32 %a21, i32* %el3
  %el122 = load i32* %el1
  %el223 = load i32* %el2
  %tmp24 = add i32 %el122, %el223
  %el325 = load i32* %el3
  %tmp26 = add i32 %tmp24, %el325
  %tmp27 = sdiv i32 %tmp26, 3
  store i32 %tmp27, i32* %avg
  %x28 = load i32** %x1
  %avg29 = load i32* %avg
  store i32 %avg29, i32* %x28
  %x30 = getelementptr inbounds i32** %x1, i32 0
```

```
    %x31 = load i32** %x30
    %x32 = getelementptr inbounds i32* %x31, i32 1
    store i32* %x32, i32** %x1
    %x33 = load i32** %x1
    %avg34 = load i32* %avg
    store i32 %avg34, i32* %x33
    %x35 = getelementptr inbounds i32** %x1, i32 0
    %x36 = load i32** %x35
    %x37 = getelementptr inbounds i32* %x36, i32 1
    store i32* %x37, i32** %x1
    %x38 = load i32** %x1
    %avg39 = load i32* %avg
    store i32 %avg39, i32* %x38
    %x40 = getelementptr inbounds i32** %x1, i32 0
    %x41 = load i32** %x40
    %x42 = getelementptr inbounds i32* %x41, i32 1
    store i32* %x42, i32** %x1
    %i43 = load i32* %i
    %tmp44 = add i32 %i43, 1
    store i32 %tmp44, i32* %i
    br label %while

  merge:                                            ; preds = %while
    ret void
  }
```

6.3 Test Automation

Our testing automation program can be invoked by the command `./testall.sh.` When the bash script `./testall.sh` is called, each file that ends with the extension “. mxl” is run and then compared with its corresponding “.out” file, if it is a test with the prefix “test,” or “.err” file if it is a test with the prefix “fail.” The “test” files compare the output of the execution of the.xml file with the expected output in the “.out” file. If the the expected output matches the actual output, testall.sh prints out “OK.” The “fail” tests compare the output of the execution of the “.mxl” file, which should result in in error, with the error in the “.err” file. If the expected error matches the actual error, testall.sh will print out “OK.”

```
dyn-160-39-10-170:ml-llvm carolinetrimble$ ./testall.sh
-n test-add1...
OK
-n test-arith1...
OK
-n test-arith2...
OK
-n test-arith3...
OK
-n test-fib...
...
```

6.4 Test Cases

The directory, `test`, contains all of the tests. As of commit [COMMIT NAME] there are [NUMBER] commits. Many of our original tests, including the tests on if statements, for loops, and the GCD and Fibonacci tests, as well as our testall.sh script, were adapted from Professor Edwards's test plan for MicroC, which is cited in our references.

The following features in our language are tested by one or more of the tests in the test plan:

7. Lessons Learned

7.1 Alex Barkume

7.2 Jared Greene

"This was my first experience working with others on a CS assignment thus far (actually, this was my first group project in college overall). I learned a lot about management and how vital communication is when it comes to working on a group coding assignment. I tend to buckle down last minute and do assignments in long stretches of continuous time. That doesn't really work when you're working in a group. Iteration is the right way to go about projects in general, but with a group coding assignments it's really the only way to do it. Ongoing communication with group members can be the most important aspect of the project because it is difficult to know where your part fits in without understanding the progress of the other members on the team. Gaining a better understanding of how to be a part of a team in a coding assignment was, though softer, a major takeaway for me."

7.3 Kyle Jackson

"When we first started, we utilized MicroC to learn the structure of a compiler and of the LLVM language. When building a compiler beginning with someone else's code, one must first have a deep understanding of how it works before adding more. In this sense starting with MicroC as a jumping off point was both a blessing and a curse. Compiling down means one must also have a deep understanding of the code to which is being compiled. I found more often than not that tweaking the LLVM code made fixing the codegen much easier. Finally, building a compiler takes a lot of time and you will spend many hours thinking about it even when you're not working on it."

7.4 Caroline Trimble

"This project differed from any other computer science project I have ever been assigned purely based on scale. Unlike most programming projects, which have about a two-week time scale, this project's semester long timeline was daunting. This long term project differed from my various short term project in two major ways: the need for constant, detailed planning and the multiple changes to that plan that occurred throughout the semester. Our project went through many major changes and it was definitely interesting and challenging to work on a product that ended up so differently than we originally planned. However, I learned that major changes are an unavoidable part of any long term project. Furthermore, to echo many other group members, communication is key and constant communications and updates from all group members are extremely important for the success of a project of this scale."

7.5 Jessica Valarezo

“Communication is key. It can be anything from letting your team know when you have free time to work on the project, to asking them for help when you don’t understand something. If you’re working on a particular feature with someone, make design decisions together. Sometimes a task may seem daunting, or a problem impossible to fix, but don’t let it discourage you.

Everyone will always have other work to do, so having a structured, in-person meeting time kept us on pace and up-to-date with everyone’s progress. In the end it will be multiple separate pieces coming together, so having an understanding of what others are working on helped put together the final product. ”

8. Future Plans

Had we had more time to work on the compiler, we would hope to implement some additional things, some of which we had stated that we planned to implement in our original proposal:

- *Parallelism*: Had we had a few more weeks to work on the compiler, our group would have loved to implement parallelism in MicroC. Having originally planned on implementing this as part of our compiler, this would make image processing in ML much faster.
- *More Python-like Syntax:* Additionally, had we had more time, we would have liked to move away from the C-like syntax of our program to a more python like syntax, allowing the user to write simpler, cleaner code.
- *Extended Image Format Capabilities*: As of now, ML only accepts the P3 (ASCII) format of the PPM image type. Our hope would be to create the capabilities for ML to accept more common file types like .JPG or .PNG and still be able to convert them into a matrix declaration, like we do currently. Although the PPM file is very simple, it is unnecessarily large, uncommon, and must be opened with a special editor.
- *Extended Standard Library*: Lastly, with more time we would have implemented a more extensive standard library that provides even more matrix and image processing capabilities.

9. Appendix

9.1 Code Listing

9.1.1 prep.ml

```
let process filename =
    let rec read_file filename =
      let file_regex = Str.regexp "<.+\\.mxl"  in
      let inc_regex = Str.regexp "#include[ ]+<.+>;" in
      let has_file l =
        let has_inc l =
          try ignore (Str.search_forward inc_regex l 0); true
          with Not_found -> false
        in
        if has_inc l then
          try ignore(Str.search_forward file_regex l 0); true
          with Not_found -> false
        else false
      in
    let lines = ref [] in
    let ic = open_in filename in
    try
      while true; do
        let l = input_line ic in
        let l = (if (has_file l) then ( Str.replace_first inc_regex
        (read_file (Str.string_after (Str.matched_string l) 1) )l ) else l) in
        lines:=  l :: !lines;
      done;
      String.concat "" (List.map (fun i -> i ^ String.make 1 '\n') (List.rev !lines))
    with End_of_file -> ignore(close_in ic);
    String.concat "" (List.map (fun i -> i ^ String.make 1 '\n') (List.rev !lines))
  in
  let process_string l =
    let read_ppm ppmname id=
      let is_decimal e =
        try ignore(Str.search_forward (Str.regexp "[0-9]+") e 0); true
```

```
       with Not_found -> false
     in
     let clean_string l =
       let only_digits = List.find_all is_decimal (Str.split (Str.regexp " ") l ) in
       (String.concat "," (List.mapi (fun i x -> if (i + 1) mod 3 = 0 then (x ^ ")")
                                             else if i mod 3 = 0 then ("(" ^ x) else x) only_digits))
     in
     let ppm_chan = open_in ppmname in
     let w = ref "" in
     let h = ref "" in
     let matrix_decl = ref "" in
     let ppm_lines = ref [] in
     try
       let _ = input_line ppm_chan in
       let d = Str.split (Str.regexp " ") (input_line ppm_chan) in
       w := List.nth d 0;
       h := List.nth d 1;
       matrix_decl := "[3][" ^ !w ^ ":" ^ !h  ^ "]";
       let _ = input_line ppm_chan in
       while true; do
         ppm_lines := (clean_string (input_line ppm_chan)) :: !ppm_lines;
       done;
       "int" ^ !matrix_decl ^ " " ^ id ^ (String.make 1 '\n') ^ ";" ^id ^ "=[|" ^ (String.concat "|"
       (List.map (fun i -> i) (List.rev !ppm_lines))) ^ "|];"
     with End_of_file -> ignore(close_in ppm_chan); "int" ^ !matrix_decl ^ " " ^ id
     ^ ";" ^ (String.make 1 '\n') ^ id ^ " = [|" ^ (String.concat "|"
     (List.map (fun i -> i) (List.rev !ppm_lines))) ^ "|];"
   in
   let file_regex = Str.regexp "\".+\\.ppm" in
  let decl_regex = Str.regexp "[0-9A-Za-z_]+[ ]*=[ ]*open\\(.+*\\);" in
  let id_regex = Str.regexp "[0-9A-Za-z_]+[ ]*=[ ]*open" in
  let has_decl l = try ignore(Str.search_forward decl_regex l 0); true
      with Not_found -> false
  in
  let get_id l = try ignore(Str.search_forward id_regex l 0);
  Str.global_replace (Str.regexp "=[ ]*open") "" (Str.matched_string l)
      with Not_found -> l
  in
  let get_file l = try ignore(Str.search_forward file_regex l 0);
  (Str.string_after (Str.matched_string l) 1)
      with Not_found -> l
  in
  let matrix_string l =
    let decl_line = Str.matched_string l in
    let ppm_file = get_file decl_line in
    let id = get_id l in
    Str.replace_first decl_regex (read_ppm ppm_file id) l
  in
  let line= (if (has_decl l) then (matrix_string l) else l) in line
in process_string (read_file filename)
```

9.1.2 scanner.mll

```
(* Ocamllex scanner for ML *)

{

  open Parser

  let un_esc s =
    Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)

}

let whitespace = [' ' '\t' '\r' '\n']
let esc = '\\' ['\\' ''' '"' 'n' 'r' 't']
let esc_ch = ''' (esc) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digits = ['0'-'9']
let alphabet = ['a'-'z' 'A'-'Z']
let alphanumund = alphabet | digits | '_'
let integer = digits+
let decimal = ['.']
let float = digits* decimal digits+ | digits+ decimal digits*
let string = '"' ( (ascii | esc)* as s ) '"'
let char = ''' ( ascii | digits ) '''
let id = alphabet alphanumund*

rule token = parse
 whitespace { token lexbuf }
| "/*"       { comment lexbuf }          (* Comments *)
| "//"       { slcomment lexbuf }        (* Single line comment *)
| '('        { LPAREN }
| ')'        { RPAREN }
| '{'        { LBRACE }
| '}'        { RBRACE }
| '['        { LBRACK }
| ']'        { RBRACK }
| '|'        { BAR }
| ';'        { SEMI }
| ':'            { COLON }
| ','        { COMMA }
| '.'        { PERIOD }
| '+'        { PLUS }
| '-'        { MINUS }
| '*'        { TIMES }
| '/'        { DIVIDE }
| '@'        { AT }
| '$'        { DOLLAR }
| "true"     { TRUE }
| "false"    { FALSE }
| '='        { ASSIGN }
| "=="       { EQ }
| "!="       { NEQ }
| '<'        { LT }
| "<="       { LEQ }
| ">"        { GT }
| ">="       { GEQ }
| "&&"       { AND }
| "||"       { OR }
| "!"        { NOT }
| "if"       { IF }
| "else"     { ELSE }
| "for"      { FOR }
| "while"    { WHILE }
| "return"   { RETURN }
| "int"      { INT }
| "float"    { FLOAT }
| "bool"     { BOOL }
| "char"     { CHAR }
| "void"     { VOID }
| "length"   { LENGTH }
| "rows"     { ROWS }
| "columns"  { COLUMNS }
| "free"     { FREE }
| integer as lxm { INTLIT(int_of_string lxm) }
| float   as lxm { FLOATLIT(float_of_string lxm) }
| string         { STRINGLIT(un_esc s) }
| char    as lxm { CHARLIT(String.get lxm 1) }
| esc_ch  as lxm { CHARLIT(String.get (un_esc lxm) 1) }
| id      as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("Illegal character " ^ Char.escaped char)) }

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

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

9.1.3 parser.mly

```
/* Ocamlyacc parser ML */

%{ open Ast %}

%token SEMI COLON LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK BAR PERIOD COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN AT DOLLAR
%token EQ NEQ LT LEQ GT GEQ AND OR NOT
%token RETURN IF ELSE ELSEIF FOR WHILE
%token TRUE FALSE
%token INT FLOAT BOOL CHAR VOID
%token LENGTH ROWS COLUMNS FREE
%token <int> INTLIT
%token <float> FLOATLIT
%token <char> CHARLIT
%token <string> STRINGLIT
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { (fst $1, snd $1) }

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

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

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

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

datatype:
    primitive                   { DataType($1) }
  | tuple_type                  { $1 }
  | matrix_type                 { $1 }
  | tuple_pointer_type          { $1 }
  | matrix_pointer_type         { $1 }
  | matrix_tuple_pointer_type { $1 }

tuple_type:
  primitive LBRACK INTLIT RBRACK { TupleType($1, $3) }

matrix_type:
    primitive LBRACK INTLIT COLON INTLIT RBRACK  { MatrixType(DataType($1), $3, $5) }
  | tuple_type LBRACK INTLIT COLON INTLIT RBRACK { MatrixType($1, $3, $5) }

tuple_pointer_type:
  primitive LBRACK RBRACK { TuplePointerType($1) }

matrix_pointer_type:
  primitive LBRACK RBRACK LBRACK RBRACK { MatrixPointerType($1)}

matrix_tuple_pointer_type:
  primitive LBRACK RBRACK LBRACK RBRACK LBRACK RBRACK { MatrixTuplePointerType($1) }

primitive:
    INT    { Int }
  | FLOAT  { Float }
  | CHAR   { Char }
  | BOOL   { Bool }
  | VOID   { Void }

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

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

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

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

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

        expr:
            literals                         { $1 }
          | ID                               { Id($1) }
          | expr PLUS   expr                 { Binop($1, Add,   $3) }
          | expr MINUS  expr                 { Binop($1, Sub,   $3) }
          | expr TIMES  expr                 { Binop($1, Mult,  $3) }
          | expr DIVIDE expr                 { Binop($1, Div,   $3) }
          | expr EQ     expr                 { Binop($1, Equal, $3) }
          | expr NEQ    expr                 { Binop($1, Neq,   $3) }
          | expr LT     expr                 { Binop($1, Less,  $3) }
          | expr LEQ    expr                 { Binop($1, Leq,   $3) }
          | expr GT     expr                 { Binop($1, Greater, $3) }
          | expr GEQ    expr                 { Binop($1, Geq,   $3) }
          | expr AND    expr                 { Binop($1, And,   $3) }
          | expr OR     expr                 { Binop($1, Or,    $3) }
          | MINUS expr %prec NEG             { Unop(Neg, $2) }
          | NOT expr                         { Unop(Not, $2) }
          | expr ASSIGN expr                 { Assign($1, $3) }
          | ID LPAREN actuals_opt RPAREN     { Call($1, $3) }
          | LPAREN expr RPAREN               { $2 }
          | ID LBRACK expr RBRACK            { TupleAccess($1, $3)}
          | ID LBRACK expr COLON expr RBRACK { MatrixAccess($1, $3, $5)}
          | ID PERIOD LENGTH                 { Length($1) }
          | ID PERIOD ROWS                   { Rows($1) }
          | ID PERIOD COLUMNS                { Columns($1) }
          | AT ID                            { TupleReference($2) }
          | DOLLAR ID                        { Dereference($2) }
          | AT AT ID                         { MatrixReference($3)}
          | AT AT AT ID                           { MatrixTupleReference($4) }
          | ID PERIOD PLUS                        { PointerIncrement($1) }
          | FREE LPAREN ID RPAREN                 { Free($3) }

        primitives:
            INTLIT    { IntLit($1) }
          | FLOATLIT  { FloatLit($1) }
          | STRINGLIT { StrLit($1) }
          | TRUE      { BoolLit(true) }
          | FALSE     { BoolLit(false) }
          | CHARLIT   { CharLit($1) }

        literals:
            primitives                                                 { $1 }
          | LBRACK array_literal
RBRACK                                  { TupleLiteral(List.rev $2) }
          | LBRACK BAR multiple_matrix BAR RBRACK      { MatrixLiteral(List.rev $3) }
          | LBRACK BAR tuple_multiple_matrix BAR RBRACK { MatrixLiteral(List.rev $3) }

        multiple_matrix:
            | array_literal {[$1]}
            | multiple_matrix BAR array_literal {$3 :: $1}

        tuple_multiple_matrix:
            | tuple_literal_list {[$1]}
            | tuple_multiple_matrix BAR tuple_literal_list {$3 :: $1}


        tuple_literal_list:
            tuple_literal                         { [$1] }
          | tuple_literal_list COMMA tuple_literal { $3 :: $1 }


        tuple_literal:
          LPAREN array_literal RPAREN { TupleLiteral(List.rev $2) }

```

```
       array_literal:
           literals                     { [$1] }
         | array_literal COMMA literals { $3 :: $1 }

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

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

9.1.4 ast.ml

```
(* Abstract-syntax Tree for ML *)

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

type uop = Neg | Not

type primitive = Int | Float | String | Bool | Char | Void

type datatype =
   TupleType of primitive * int
 | MatrixType of datatype * int * int
 | DataType of primitive
 | TuplePointerType of primitive
 | MatrixPointerType of primitive
 | MatrixTuplePointerType of primitive

type var_dec = datatype * string

type expr =
    IntLit of int
```

```
  | CharLit of char
  | FloatLit of float
  | StrLit of string
  | BoolLit of bool
  | TupleAccess of string * expr
  | MatrixAccess of string * expr * expr
  | PointerIncrement of string
  | TupleLiteral of expr list
  | MatrixLiteral of expr list list
  | Length of string
  | Rows of string
  | Columns of string
  | Free of string
  | TupleReference of string
  | Dereference of string
  | MatrixReference of string
  | MatrixTupleReference of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of string * expr list
  | Noexpr

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

type func_decl = {
    datatype : datatype;
    fname : string;
    formals : var_dec list;
    locals : var_dec list;
    body : stmt list;
  }

type program = var_dec list * func_decl list

(* Pretty-printing functions *)

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

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

let string_of_tuple t =
  let rec string_of_tuple_literal = function
      [] -> "]"
    | [hd] -> (match hd with
                IntLit(i) -> string_of_int i
              | FloatLit(f) -> string_of_float f
              | StrLit(s) -> s
              | CharLit(c) -> String.make 1 c
              | BoolLit(true) -> "true"
              | BoolLit(false) -> "false"
```

```
          | Id(s) -> s
          | _ -> raise( Failure("Illegal expression in tuple primitive") )) ^ string_of_tuple_literal []
| hd::tl -> (match hd with
                IntLit(i) -> string_of_int i ^ ", "
              | FloatLit(f) -> string_of_float f ^ ", "
              | StrLit(s) -> s ^ ", "
                      | CharLit(c) -> (String.make 1 c) ^ ", "
                      | BoolLit(true) -> "true" ^ ", "
                      | BoolLit(false) -> "false" ^ ", "
                      | Id(s) -> s
                      | _ -> raise( Failure("Illegal expression in tuple primitive") )) ^ string_of_tuple_literal tl
      in
      "[" ^ string_of_tuple_literal t

    let string_of_matrix m r c =
      let rec string_of_matrix_literal = function
          [] -> "| " ^ string_of_int r ^ ", " ^ string_of_int c ^ "]"
        | [hd] -> (match hd with
                    IntLit(i) -> string_of_int i
                  | FloatLit(f) -> string_of_float f
                  | StrLit(s) -> s
                  | CharLit(c) -> String.make 1 c
                  | BoolLit(true) -> "true"
                  | BoolLit(false) -> "false"
                  | Id(s) -> s
                  | TupleLiteral(t) -> string_of_tuple t
                  | _ -> raise( Failure("Illegal expression in matrix primitive") )) ^ string_of_matrix_literal []
        | hd::tl -> (match hd with
                        IntLit(i) -> string_of_int i ^ ", "
                      | FloatLit(f) -> string_of_float f ^ ", "
                      | StrLit(s) -> s ^ ", "
                      | CharLit(c) -> (String.make 1 c) ^ ", "
                      | BoolLit(true) -> "true" ^ ", "
```

```
             | BoolLit(false) -> "false" ^ ", "
             | Id(s) -> s
             | TupleLiteral(t) -> string_of_tuple t ^ ", "
             | _ -> raise( Failure("Illegal expression in matrix primitive") )) ^ string_of_matrix_literal tl
     in
     "[|" ^ string_of_matrix_literal m

    let rec string_of_expr = function
      IntLit(i) -> string_of_int i
     | FloatLit(f) -> string_of_float f
     | StrLit(s) -> s
     | CharLit(c) -> String.make 1 c
     | BoolLit(true) -> "true"
     | BoolLit(false) -> "false"
     | Id(s) -> s
     | TupleLiteral(t) -> string_of_tuple t
     | MatrixLiteral(_) -> "matrix literal"
     | TupleAccess(s, e) -> s ^ "[" ^ string_of_expr e ^ "]"
     | MatrixAccess(s, e1, e2) -> s ^ "[" ^ string_of_expr e1 ^ ":" ^ string_of_expr e2 ^ "]"
     | PointerIncrement(s) -> s ^ "++"
     | Length(s) -> s ^ "." ^ "length"
     | Rows(s) -> s ^ "." ^ "rows"
     | Columns(s) -> s ^ "." ^ "columns"
     | Free(s) -> "free" ^ "(" ^ s ^ ")"
     | TupleReference(s) -> "@" ^ s
     | Dereference(s) -> "$" ^ s
     | MatrixReference(s) -> "@@" ^ s
     | MatrixTupleReference(s) -> "@@@" ^ s
     | Binop(e1, o, e2) ->
       string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
     | Unop(o, e) -> string_of_uop o ^ string_of_expr e
     | Assign(e1, e2) -> string_of_expr e1 ^ " = " ^ string_of_expr e2
     | Call(f, el) ->
       f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
```

```
  | Noexpr -> ""

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

let string_of_typ = function
    DataType(Int) -> "int"
  | DataType(Float) -> "float"
  | DataType(String) -> "str"
  | DataType(Char) -> "char"
  | DataType(Bool) -> "bool"
  | DataType(Void) -> "void"
  | TupleType(p, l) -> (match p with
                          Int -> "int" ^ "[" ^ string_of_int l ^ "]"
                        | Float -> "float" ^ "[" ^ string_of_int l ^ "]"
                        | Char -> "char" ^ "[" ^ string_of_int l ^ "]"
                        | String -> "str" ^ "[" ^ string_of_int l ^ "]"
                        | Bool -> "bool" ^ "[" ^ string_of_int l ^ "]"
                        | Void -> "void" ^ "[" ^ string_of_int l ^ "]")
  | MatrixType(t, l1, l2) -> (match t with
                                DataType(Int) -> "int" ^ "[" ^ string_of_int l1 ^ ":" ^ string_of_int l2 ^ "]"
                              | DataType(Float) -> "float" ^ "[" ^ string_of_int l1 ^ ":" ^ string_of_int l2 ^ "]"
                              | DataType(Char) -> "char" ^ "[" ^ string_of_int l1 ^ ":" ^ string_of_int l2 ^ "]"
```

```
                                      | DataType(String) -
> "str" ^ "[" ^ string_of_int l1 ^ ":" ^ string_of_int l2 ^ "]"
                                      | DataType(Bool) -
> "bool" ^ "[" ^ string_of_int l1 ^ ":" ^ string_of_int l2 ^ "]"
                                      | DataType(Void) -
> "void" ^ "[" ^ string_of_int l1 ^ ":" ^ string_of_int l2 ^ "]"
                                      | TupleType(p, l) ->  (match p with
                                                                Int -
> "int" ^ "[" ^ string_of_int l ^ "]" ^ "[" ^ string_of_int l1 ^ ":"^ string_of_int l2 ^ "]"
                                                              | Float -
> "float" ^ "[" ^ string_of_int l ^ "]" ^ "[" ^ string_of_int l1 ^":" ^ string_of_int l2 ^ "]"
                                                              | Char -
> "char" ^ "[" ^ string_of_int l ^ "]" ^ "[" ^ string_of_int l1 ^":" ^ string_of_int l2 ^ "]"
                                                              | String -
> "str" ^ "[" ^ string_of_int l ^ "]" ^ "[" ^ string_of_int l1 ^":" ^ string_of_int l2 ^ "]"
                                                              | Bool -
> "bool" ^ "[" ^ string_of_int l ^ "]" ^ "[" ^ string_of_int l1 ^":" ^ string_of_int l2 ^ "]"
                                                              | Void -
> "void" ^ "[" ^ string_of_int l ^ "]" ^ "[" ^ string_of_int l1 ^":" ^ string_of_int l2 ^ "]"
                                                            )
                                      | _ -> raise ( Failure ("Illegal matrix of matrices") )
                                    )
          | TuplePointerType(Int) -> "int[]"
          | TuplePointerType(Float) -> "float[]"
          | TuplePointerType(String) -> "str[]"
          | TuplePointerType(Char) -> "char[]"
          | TuplePointerType(Bool) -> "bool[]"
          | TuplePointerType(Void) -> "void[]"
          | MatrixPointerType(Int) -> "int[][]"
          | MatrixPointerType(Float) -> "float[][]"
          | MatrixPointerType(Char) -> "char[][]"
          | MatrixPointerType(String) -> "str[][]"
          | MatrixPointerType(Bool) -> "bool[][]"
          | MatrixPointerType(Void) -> "void[][]"
          | MatrixTuplePointerType(Int) -> "int[][][]"
          | MatrixTuplePointerType(Float) -> "float[][][]"
```

```
        | MatrixTuplePointerType(Char) -> "char[][][]"
        | MatrixTuplePointerType(String) -> "str[][][]"
        | MatrixTuplePointerType(Bool) -> "bool[][][]"
        | MatrixTuplePointerType(Void) -> "void[][][]"

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

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

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

9.1.5 pretty.ml

```
open Ast

let _ =
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
  print_endline (string_of_program program)
```

9.1.6 semant.ml

```
* Semantic checking for the ML compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.
   Check each global variable, then check each function *)

let check (globals, functions) =

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

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

  let check_assign lvaluet rvaluet err =
    match (lvaluet, rvaluet) with
      (DataType(Int), DataType(Int)) -> lvaluet
    | (DataType(Float), DataType(Float)) -> lvaluet
    | (DataType(Char), DataType(Char)) -> lvaluet
    | (DataType(String), DataType(String)) -> lvaluet
    | (DataType(Bool), DataType(Bool)) -> lvaluet
    | (DataType(Void), DataType(Void)) -> lvaluet
    | (TupleType(Int, l1), TupleType(Int, l2)) -> if l1 == l2 then lvaluet else if l1 == 0 then lvaluet else raise err
    | (TupleType(Float, l1), TupleType(Float, l2)) -> if l1 == l2 then lvaluet else if l1 == 0 then lvaluet else raise err
    | (TupleType(Char, l1), TupleType(Char, l2)) -> if l1 == l2 then lvaluet else if l1 == 0 then lvaluet else raise err
    | (TupleType(Bool, l1), TupleType(Bool, l2)) -> if l1 == l2 then lvaluet else if l1 == 0 then lvaluet else raise err
    | (MatrixType(DataType(Int), r1, c1), MatrixType(DataType(Int), r2, c2)) -
> if r1 == r2 && c1 == c2 then lvaluet else raiseerr
    | (MatrixType(DataType(Float), r1, c1), MatrixType(DataType(Float), r2, c2)) -
> if r1 == r2 && c1 == c2 then lvaluet elseraise err
    | (MatrixType(TupleType(Int, d1), r1, c1), MatrixType(TupleType(Int, d2), r2, c2)) -
> if d1 == d2 && r1 == r2 && c1 == c2then lvaluet else raise err
    | (MatrixType(TupleType(Float, d1), r1, c1), MatrixType(TupleType(Float, d2), r2, c2)) -
> if d1 == d2 && r1 == r2 && c1 == c2then lvaluet else raise err
    | (TuplePointerType(Int), TuplePointerType(Int)) -> lvaluet
    | (TuplePointerType(Float), TuplePointerType(Float)) -> lvaluet
    | (TuplePointerType(Char), TuplePointerType(Char)) -> lvaluet
    | (TuplePointerType(Bool), TuplePointerType(Bool)) -> lvaluet
    | (MatrixPointerType(Int), MatrixPointerType(Int)) -> lvaluet
    | (MatrixPointerType(Float), MatrixPointerType(Float)) -> lvaluet
    | (MatrixTuplePointerType(Int), MatrixTuplePointerType(Int)) -> lvaluet
    | (MatrixTuplePointerType(Float), MatrixTuplePointerType(Float)) -> lvaluet
    | _ -> raise err
in

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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

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

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

(* Function declaration for a named function *)
        let built_in_decls =  StringMap.add "print"
        { datatype = DataType(Void); fname = "print"; formals = [(DataType(Int), "x")];
        locals = []; body = [] } (StringMap.add "printsl"
          { datatype = DataType(Void); fname = "printsl"; formals = [(DataType(Int), "x")];
          locals = []; body = [] } (StringMap.add "printb"
          { datatype = DataType(Void); fname = "printb"; formals = [(DataType(Bool), "x")];
          locals = []; body = [] } (StringMap.add "printbsl"
          { datatype = DataType(Void); fname = "printbsl"; formals = [(DataType(Bool), "x")];
```

```
            locals = []; body = [] } (StringMap.add "prints"
              { datatype = DataType(Void); fname = "prints"; formals = [(DataType(String), "s")
];
              locals = []; body = [] } (StringMap.add "printssl"
                { datatype = DataType(Void); fname = "printssl"; formals = [(DataType(String),
"s")];
                locals = []; body = [] } (StringMap.add "printf"
                { datatype = DataType(Void); fname = "printf"; formals = [(DataType(Float), "x"
)];
              locals = []; body = [] } (StringMap.add "printfsl"
              { datatype = DataType(Void); fname = "printfsl"; formals = [(DataType(Float), "x"
)];
            locals = []; body = [] } (StringMap.add "printc"
            { datatype = DataType(Void); fname = "printc"; formals = [(DataType(Char), "x")];
          locals = []; body = [] } (StringMap.add "printcsl"
          { datatype = DataType(Void); fname = "printcsl"; formals = [(DataType(Char), "x")];
        locals = []; body = [] } (StringMap.singleton "open"
                { datatype = DataType(Void); fname = "open"; formals= [(DataType(String), "s")]
;
              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 check_function func =

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

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

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

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

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

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

          let type_of_tuple t =
            match (List.hd t) with
              IntLit _ -> TupleType(Int, List.length t)
            | FloatLit _ -> TupleType(Float, List.length t)
            | CharLit _ -> TupleType(Char, List.length t)
            | BoolLit _ -> TupleType(Bool, List.length t)
            | _ -> raise (Failure ("illegal tuple type")) in

          let rec check_tuple_literal tt l i =
            let length = List.length l in
            match (tt, List.nth l i) with
              (TupleType(Int, _), IntLit _) -> if i == length - 1 then TupleType(Int,
length) else check_tuple_literal (TupleType(Int, length)) l (succ i)
            | (TupleType(Float, _), FloatLit _) -> if i == length - 1 then TupleType(Float,
length) else check_tuple_literal(TupleType(Float, length)) l (succ i)
```

```
            | (TupleType(Char, _), CharLit _) -> if i == length - 1 then TupleType(Char, length) else check_tuple_literal(TupleType(Char, length)) l (succ i)
            | (TupleType(Bool, _), BoolLit _) -> if i == length - 1 then TupleType(Bool, length) else check_tuple_literal(TupleType(Bool, length)) l (succ i)
            | _ -> raise (Failure ("illegal tuple literal"))
          in

          let access_type = function
              TupleType(p, _) -> DataType(p)
            | _ -> raise (Failure ("illegal access type")) in

          let matrix_acces_type = function
              MatrixType(t, _, _) -> t
            | _ -> raise (Failure ("illegal matrix access") ) in

          let type_of_matrix m r c =
            match (List.hd (List.hd m)) with
                IntLit _ -> MatrixType(DataType(Int), r, c)
              | FloatLit _ -> MatrixType(DataType(Float), r, c)
              | TupleLiteral t -> MatrixType((type_of_tuple) t, r, c)
              | _ -> raise (Failure ("illegal matrix type"))
          in

          let check_pointer_type = function
              TuplePointerType(t) -> TuplePointerType(t)
            | MatrixPointerType(t) -> MatrixPointerType(t)
            | MatrixTuplePointerType(t) -> MatrixTuplePointerType(t)
            | _ -> raise ( Failure ("cannot increment a non-pointer type") )
          in

          let check_tuple_pointer_type = function
              TupleType(p, _) -> TuplePointerType(p)
            | _ -> raise ( Failure ("cannot reference a non-tuple pointer type"))
          in

          let check_matrix_pointer_type = function
```

```
            MatrixType(DataType(p), _, _) -> MatrixPointerType(p)
          | _ -> raise ( Failure ("cannot reference a non-matrix pointer type"))
        in

        let check_matrix_tuple_pointer_type = function
            MatrixType(TupleType(p, _), _, _) -> MatrixTuplePointerType(p)
          | _ -> raise ( Failure ("cannot reference a non-matrix-tuple pointer type"))
        in

        let pointer_type = function
          | TuplePointerType(p) -> DataType(p)
          | MatrixPointerType(p) -> DataType(p)
          | MatrixTuplePointerType(p) -> DataType(p)
          | _ -> raise ( Failure ("cannot dereference a non-pointer type") ) in

        (* Return the type of an expression or throw an exception *)
        let rec expr = function
          IntLit _ -> DataType(Int)
        | FloatLit _ -> DataType(Float)
        | CharLit _ -> DataType(Char)
        | StrLit _ -> DataType(String)
        | BoolLit _ -> DataType(Bool)
        | Id s -> type_of_identifier s
        | TupleLiteral t -> check_tuple_literal (type_of_tuple t) t 0
        | MatrixLiteral m -> type_of_matrix m (List.length m) (List.length (List.hd m))
        | TupleAccess(s, e) -> let _ = (match (expr e) with
                                          DataType(Int) -> DataType(Int)
                                        | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                               access_type (type_of_identifier s)
        | MatrixAccess(s, e1, e2) -> let _ = (match (expr e1) with
                                                DataType(Int) -> DataType(Int)
                                              | _ -> raise (Failure ("attempting to access with a non-integer type")))
                                     and _ = (match (expr e2) with
                                                DataType(Int) -> DataType(Int)
```

```
                        | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                    matrix_acces_type (type_of_identifier s)
     | PointerIncrement(s) -> check_pointer_type (type_of_identifier s)
     | Length(s) -> (match (type_of_identifier s) with
              TupleType(_, _) -> DataType(Int)
             | MatrixType(TupleType(_, _), _, _) -> DataType(Int)
             | _ -> raise(Failure ("cannot get the length of non-matrix-of-tuples or non-tuple datatype")))
     | Rows(s) -> (match (type_of_identifier s) with
             MatrixType(_, _, _) -> DataType(Int)
            | _ -> raise (Failure ("cannot get the rows of non-matrix datatype")))
     | Columns(s) -> (match (type_of_identifier s) with
               MatrixType(_, _, _) -> DataType(Int)
              | _ -> raise (Failure ("cannot get the rows of non-matrix datatype")))
     | Free(s) -> (match (type_of_identifier s) with
             MatrixType(TupleType(_, _), _, _) -> DataType(Void)
            | _ -> raise (Failure ("cannot free a non-matrix-tuple type")))
     | TupleReference(s) -> check_tuple_pointer_type (type_of_identifier s)
     | Dereference(s) -> pointer_type (type_of_identifier s)
     | MatrixReference(s) -> check_matrix_pointer_type (type_of_identifier s)
     | MatrixTupleReference(s) -> check_matrix_tuple_pointer_type (type_of_identifier s)
     | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
     (match op with
       Add | Sub | Mult | Div when t1 = DataType(Int) && t2 = DataType(Int) -> DataType(Int)
      | Add | Sub | Mult | Div when t1 = DataType(Float) && t2 = DataType(Float) -> DataType(Float)
      | Equal | Neq when t1 = t2 -> DataType(Bool)
      | Less | Leq | Greater | Geq when t1 = DataType(Int) && t2 = DataType(Int) -> DataType(Bool)
      | Less | Leq | Greater | Geq when t1 = DataType(Float) && t2 = DataType(Float) -> DataType(Float)
      | And | Or when t1 = DataType(Bool) && t2 = DataType(Bool) -> DataType(Bool)
      | _ -> raise (Failure ("illegal binary operator " ^
              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
              string_of_typ t2 ^ " in " ^ string_of_expr e))
          )
          | Unop(op, e) as ex -> let t = expr e in
          (match op with
              Neg when t = DataType(Int) -> DataType(Int)
            | Neg when t = DataType(Float) -> DataType(Float)
            | Not when t = DataType(Bool) -> DataType(Bool)
            | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
              string_of_typ t ^ " in " ^ string_of_expr ex)))
          | Noexpr -> DataType(Void)
          | Assign(e1, e2) as ex -> let lt = (match e1 with
                                                TupleAccess(s, e) -> (match (expr e) with
                                                                        DataType(Int) -> (match (type_of_identifier s) with
                                                                                            TupleType(p, _) -> (match p with
                 Int -> DataType(Int)
               | Float ->DataType(Float)
               | Char -> DataType(Char)
               | String ->DataType(String)
               | Bool -> DataType(Bool)
               | Void -> DataType(Void)
             )
                                                                                          | _ -> raise ( Failure ("cannot access a non-tuple type") )
                                                                                        )
                                                                      | _ -> raise ( Failure ("expression is not of type int") )
```

```
                                                            )
                                      | MatrixAccess(s, _, _) -
> (match (type_of_identifier s) with
                                                                MatrixType(t, _,
_) -> (match t with
         DataType(Int) ->DataType(Int)
       | DataType(Float) ->DataType(Float)
       | TupleType(p, l) ->TupleType(p, l)
       | _ -> raise ( Failure("illegal matrix of matrices") )
      )
                                                                | _ -
> raise ( Failure ("cannot access a primitive") )
                                                            )
                                      | _ -> expr e1)
                               and rt = (match e2 with
                                        TupleAccess(s, e) -> (match (expr e) with
                                                          DataType(Int) -
> (match (type_of_identifier s) with
                                                                          Tup
leType(p, _) -> (match p with
            Int -> DataType(Int)
          | Float ->DataType(Float)
          | Char -> DataType(Char)
          | String ->DataType(String)
          | Bool -> DataType(Bool)
```

```
                | Void -> DataType(Void)
              )
                                                                                 | _
    -> raise ( Failure ("cannot access a non-tuple type") )
                                                                               )
                                                                  | _ -
    > raise ( Failure ("expression is not of datatype int") )
                                                                )
                                              | MatrixAccess(s, _, _) -
    > (match (type_of_identifier s) with
                                                                       MatrixType(t, _,
    _) -> (match t with
          DataType(Int) -> DataType(Int)
        | DataType(Float) ->DataType(Float)
        | TupleType(p, l) -> TupleType(p, l)
        | _ -> raise ( Failure ("illegal matrix of matrices") )
      )
                                                                       | _ -
    > raise ( Failure ("cannot access a primitive") )
                                                                     )
                                              | _ -> expr e2) in
          check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
            " = " ^ string_of_typ rt ^ " in " ^
          string_of_expr ex))
          | Call(fname, actuals) as call -> let fd = function_decl fname in
          if List.length actuals != List.length fd.formals then
           raise (Failure ("expecting " ^ string_of_int
             (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
          else
```

```
 List.iter2 (fun (ft, _) e -> let et = expr e in
  ignore (check_assign ft et
    (Failure ("illegal actual argument found " ^ string_of_typ et ^
      " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
 fd.formals actuals;
 fd.datatype
in

let check_bool_expr e =
  match (expr e) with
    DataType(Bool) -> ()
  | _ -> raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
in

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

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

stmt (Block func.body)

```

```
        in
        List.iter check_function functions
```

9.1.7 exceptions.ml

```
(* Binop Exceptions *)

exception UnsupportedBinaryOperationOnTypes of string * string
exception UnsupportedBinOp

(* Unop Exceptions *)

exception UnsupportedUnaryOperationOnType of string
exception UnsupportedUnaryOperationOnFloat

(* Tuple Exceptions *)

exception UnsupportedTupleOfTuples
exception UnsupportedTupleType
exception InvalidTuplePointerType
exception InvalidUseOfLength

(* Matrix Exceptions *)

exception UnsupportedMatrixofMatrices
exception UnsupportedMatrixType
exception InvalidUseOfRows
exception InvalidUseOfColumns

(* Assignment Exceptions *)

exception IllegalAssignment

(* Pointer Exceptions *)

exception IllegalPointerType
```

```

(* Return Execeptions *)

exception IllegalReturnType
```

9.1.8 codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR
LLVM tutorial: Make sure to read the OCaml version of the tutorial
http://llvm.org/docs/tutorial/index.html
Detailed documentation on the OCaml LLVM library:
http://llvm.moe/
http://llvm.moe/ocaml/
*)
module L = Llvm
module A = Ast
open Exceptions

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "ML"
  and i32_t      = L.i32_type    context
  and float_t    = L.double_type context
  and i8_t       = L.i8_type     context
  and pointer_t = L.pointer_type
  and array_t    = L.array_type
  and i1_t       = L.i1_type     context
  and void_t     = L.void_type   context in

  let ltype_of_typ = function
      A.DataType(A.Int)      -> i32_t
    | A.DataType(A.Float)    -> float_t
    | A.DataType(A.Char)     -> i8_t
    | A.DataType(A.String)   -> pointer_t i8_t
    | A.DataType(A.Bool)     -> i1_t
    | A.DataType(A.Void)     -> void_t
    | A.TupleType(typ, size) -> (match typ with
                                   A.Int    -> array_t i32_t size
                                 | A.Float  -> array_t float_t size
                                 | A.Char   -> array_t i8_t size
                                 | A.Bool   -> array_t i1_t size
                                 | _ -> raise (UnsupportedTupleType))
    | A.MatrixType(typ, size1, size2) -> (match typ with
                                            A.DataType(A.Int) -> array_t (array_t i32_t size2) size1
                                          | A.DataType(A.Float) -> array_t (array_t float_t size2) size1
                                          | A.TupleType(typ1, size3) -> (match typ1 with
                                                                         | A.Int -> array_t (array_t (array_t i32_t size3) size2)size1
                                                                         | A.Float -> array_t (array_t (array_t float_t size3)size2) size1
                                                                         | _ -> raise (UnsupportedMatrixType)
                                                                        )
                                          | _ -> raise ( UnsupportedMatrixType )
                                         )
    | A.TuplePointerType(t) -> (match t with
                                 A.Int -> pointer_t i32_t
                               | A.Float -> pointer_t float_t
                               | A.Char -> pointer_t i8_t
                               | A.Bool -> pointer_t i1_t
                               | _ -> raise (IllegalPointerType))
    | A.MatrixPointerType(t) -> (match t with
                                   A.Int -> pointer_t i32_t
                                 | A.Float -> pointer_t float_t
                                 | _ -> raise (IllegalPointerType))
    | A.MatrixTuplePointerType(t) -> (match t with
                                         A.Int -> pointer_t i32_t
```

```
                                       | A.Float -> pointer_t float_t
                                       | _ -> raise (IllegalPointerType))
  in

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

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

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

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

  let int_sl_format_str = L.build_global_stringptr "%d" "fmt" builder
  and float_sl_format_str = L.build_global_stringptr "%f" "fmt" builder in

```

```
    let char_format_str = L.build_global_stringptr "%c\n" "fmt" builder
    and char_sl_format_str = L.build_global_stringptr "%c" "fmt" builder in

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

              let add_local m (t, n) =
            let local_var =
            (match t with
               A.MatrixType(A.TupleType(_, l), r, c) -> if r * c * l > 1000
                                                          then L.build_malloc (ltype_of_typ t) n builder
                                                        else
                                                          L.build_alloca (ltype_of_typ t) n builder
             | _ -> 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.A.formals
                  (Array.to_list (L.params the_function)) in
              List.fold_left add_local formals fdecl.A.locals in

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

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

            let type_of_identifier s =
              let symbols = check_function in
              try StringMap.find s symbols
              with Not_found -> raise (Failure ("symbol not found"))
            in

            let get_tuple_type tuple =
              match (List.hd tuple) with
                A.IntLit _ -> ltype_of_typ (A.DataType(A.Int))
              | A.FloatLit _ -> ltype_of_typ (A.DataType(A.Float))
              | A.CharLit _ -> ltype_of_typ (A.DataType(A.Char))
              | A.StrLit _ -> ltype_of_typ (A.DataType(A.String))
              | A.BoolLit _ -> ltype_of_typ (A.DataType(A.Bool))
              | _ -> raise (UnsupportedTupleType) in

            let build_matrix_access s i1 i2 i3 builder isAssign =
              if isAssign
                then L.build_gep (lookup s) [| i1; i2; i3 |] s builder
              else
                 L.build_load (L.build_gep (lookup s) [| i1; i2; i3 |] s builder) s builder
            in

            let build_tuple_access s i1 i2 builder isAssign =
              if isAssign
                then L.build_gep (lookup s) [| i1; i2 |] s builder
              else
                 L.build_load (L.build_gep (lookup s) [| i1; i2 |] s builder) s builder
            in

            let build_tuple_argument s builder =
              L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0 |] s
  builder
            in
```

```

    let build_matrix_argument s builder =
     L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0; L.const_int i32_t 0 |] s builder
    in

    let build_matrix_tuple_argument s builder =
     L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0; L.const_int i32_t 0; L.const_int i32_t 0 |] s builder
    in

    let build_pointer_dereference s builder isAssign =
     if isAssign
      then L.build_load (lookup s) s builder
     else
      L.build_load (L.build_load (lookup s) s builder) s builder
    in

    let build_pointer_increment s builder isAssign =
     if isAssign
      then L.build_load (L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 1 |] s builder) s builder
     else
      L.build_in_bounds_gep (L.build_load (L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0 |] s builder) s builder) [|L.const_int i32_t 1 |] s builder
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder = function
       A.IntLit i -> L.const_int i32_t i
     | A.FloatLit f -> L.const_float float_t f
     | A.CharLit c -> L.const_int i8_t (Char.code c)
     | A.StrLit s -> L.const_string context s
     | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
     | A.TupleLiteral t -> L.const_array (get_tuple_type t) (Array.of_list (List.map (expr builder) t))
```

```
              | A.TupleAccess(s, e) -> let i = expr builder e in build_tuple_access
s (L.const_int i32_t 0) i builder false
              | A.MatrixLiteral m -> (match (List.hd (List.hd m)) with
                                        A.FloatLit _ -> let realOrder=List.map List.rev
m in let i32Lists = List.map (List.map (expr
builder)) realOrder in let listOfArrays=List.map Array.of_list
i32Lists in let i32ListOfArrays = List.map (L.const_array
float_t)listOfArrays in let arrayOfArrays=Array.of_list
i32ListOfArrays in L.const_array (array_t float_t (List.length (List.hd m)))arrayOfArrays
                                      | A.IntLit _ -> let realOrder=List.map List.rev
m in let i32Lists = List.map (List.map (expr
builder)) realOrder in let listOfArrays=List.map Array.of_list
i32Lists in let i32ListOfArrays = List.map (L.const_array
i32_t)listOfArrays in let arrayOfArrays=Array.of_list i32ListOfArrays in L.const_array (array_t
i32_t (List.length (List.hd m)))arrayOfArrays
                                      | A.TupleLiteral t -> let realOrder=List.map List.rev
m in let i32Lists = List.map (List.map (expr
builder)) realOrder in let listOfArrays=List.map Array.of_list
i32Lists in let i32ListOfArrays = List.map (L.const_array (array_t(get_tuple_type
t) (List.length t))) listOfArrays in let arrayOfArrays=Array.of_list
i32ListOfArrays in L.const_array (array_t(array_t (get_tuple_type t) (List.length
t)) (List.length (List.hd m))) arrayOfArrays
                                      | _ -> raise ( UnsupportedMatrixType )
                                      )
              | A.MatrixAccess (s, e1, e2) -> let i1 = expr builder e1 and i2 = expr builder
e2 in build_matrix_access s (L.const_int i32_t 0) i1 i2 builder false
              | A.PointerIncrement (s) ->  build_pointer_increment s builder false
              | A.Length (s) -> (match (type_of_identifier s) with
                                   A.TupleType(_, l) -> L.const_int i32_t l
                                 | A.MatrixType(A.TupleType(_, l), _, _) -> L.const_int i32_t l
                                 | _ -> raise ( InvalidUseOfLength )
                                )
              | A.Rows(s) -> (match (type_of_identifier s) with
                                A.MatrixType(_, r, _) -> L.const_int i32_t r
                              | _ -> raise ( InvalidUseOfRows ))
              | A.Columns(s) -> (match (type_of_identifier s) with
                                   A.MatrixType(_, _, c) -> L.const_int i32_t c
                                 | _ -> raise ( InvalidUseOfColumns ))
              | A.Free(s) -> L.build_free (lookup s) builder
              | A.TupleReference (s) -> build_tuple_argument s builder
              | A.Dereference (s) -> build_pointer_dereference s builder false
              | A.MatrixReference (s) -> build_matrix_argument s builder
              | A.MatrixTupleReference (s) -> build_matrix_tuple_argument s builder
              | A.Noexpr -> L.const_int i32_t 0
              | A.Id s -> L.build_load (lookup s) s builder
              | A.Binop (e1, op, e2) ->
              let e1' = expr builder e1
              and e2' = expr builder e2 in

              let float_bops operator =
                match operator with
                  A.Add     -> L.build_fadd e1' e2' "tmp" builder
                | A.Sub     -> L.build_fsub e1' e2' "tmp" builder
                | A.Mult    -> L.build_fmul e1' e2' "tmp" builder
                | A.Div     -> L.build_fdiv e1' e2' "tmp" builder
                | A.And     -> L.build_and e1' e2' "tmp" builder
                | A.Or      -> L.build_or e1' e2' "tmp" builder
                | A.Equal   -> L.build_fcmp L.Fcmp.Oeq e1' e2' "tmp" builder
                | A.Neq     -> L.build_fcmp L.Fcmp.One e1' e2' "tmp" builder
                | A.Less    -> L.build_fcmp L.Fcmp.Olt e1' e2' "tmp" builder
                | A.Leq     -> L.build_fcmp L.Fcmp.Ole e1' e2' "tmp" builder
                | A.Greater -> L.build_fcmp L.Fcmp.Ogt e1' e2' "tmp" builder
                | A.Geq     -> L.build_fcmp L.Fcmp.Oge e1' e2' "tmp" builder
              in

              let int_bops operator =
                match operator with
                  A.Add     -> L.build_add e1' e2' "tmp" builder
                | A.Sub     -> L.build_sub e1' e2' "tmp" builder
                | A.Mult    -> L.build_mul e1' e2' "tmp" builder
                | A.Div     -> L.build_sdiv e1' e2' "tmp" builder
                | A.And     -> L.build_and e1' e2' "tmp" builder
```

```ocaml
          | A.Or      -> L.build_or e1' e2' "tmp" builder
          | A.Equal   -> L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
          | A.Neq     -> L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
          | A.Less    -> L.build_icmp L.Icmp.Slt e1' e2' "tmp" builder
          | A.Leq     -> L.build_icmp L.Icmp.Sle e1' e2' "tmp" builder
          | A.Greater -> L.build_icmp L.Icmp.Sgt e1' e2' "tmp" builder
          | A.Geq     -> L.build_icmp L.Icmp.Sge e1' e2' "tmp" builder
        in

        let string_of_e1'_llvalue = L.string_of_llvalue e1'
        and string_of_e2'_llvalue = L.string_of_llvalue e2' in

        let space = Str.regexp " " in

        let list_of_e1'_llvalue = Str.split space string_of_e1'_llvalue
        and list_of_e2'_llvalue = Str.split space string_of_e2'_llvalue in

        let i32_re = Str.regexp "i32\\|i32*\\|i8\\|i8*\\|i1\\|i1*"
        and float_re = Str.regexp "double\\|double*" in

        let rec match_string regexp str_list i =
           let length = List.length str_list in
           match (Str.string_match regexp (List.nth str_list i) 0) with
             true -> true
           | false -> if (i > length - 2) then false else match_string regexp str_list (succ i) in

        let get_type llvalue =
           match (match_string i32_re llvalue 0) with
             true  -> "int"
           | false -> (match (match_string float_re llvalue 0) with
                         true -> "float"
                       | false -> "") in

        let e1'_type = get_type list_of_e1'_llvalue
        and e2'_type = get_type list_of_e2'_llvalue in
```

```

let build_ops_with_types typ1 typ2 =
 match (typ1, typ2) with
  "int", "int" -> int_bops op
 | "float" , "float" -> float_bops op
 | _, _ -> raise(UnsupportedBinaryOperationOnTypes (typ1, typ2))
in

build_ops_with_types e1'_type e2'_type
| A.Unop(op, e) ->
  let e' = expr builder e in

 let float_uops operator =
  match operator with
   A.Neg -> L.build_fneg e' "tmp" builder
  | A.Not -> raise(UnsupportedUnaryOperationOnFloat)  in

 let int_uops operator =
  match operator with
   A.Neg -> L.build_neg e' "tmp" builder
  | A.Not -> L.build_not e' "tmp" builder in

 let string_of_e'_llvalue = L.string_of_llvalue e' in

 let space = Str.regexp " " in

 let list_of_e'_llvalue = Str.split space string_of_e'_llvalue in

 let i32_re = Str.regexp "i32\\|i32*\\|i8\\|i8*\\|i1\\|i1*"
 and float_re = Str.regexp "double\\|double*" in

 let rec match_string regexp str_list i =
   let length = List.length str_list in
   match (Str.string_match regexp (List.nth str_list i) 0) with
    true -> true
```

```
                   | false -> if (i > length - 2) then false else match_string regexp str_list (succ i) in

                let get_type llvalue =
                   match (match_string i32_re llvalue 0) with
                     true  -> "int"
                   | false -> (match (match_string float_re llvalue 0) with
                                 true -> "float"
                               | false -> "") in

                let e'_type = get_type list_of_e'_llvalue  in

                let build_ops_with_type typ =
                  match typ with
                    "int" -> int_uops op
                  | "float" -> float_uops op
                  | _ -> raise(UnsupportedUnaryOperationOnType typ)
                in

                build_ops_with_type e'_type
              | A.Assign (e1, e2) -> let e1' = (match e1 with
                                                    A.Id s -> lookup s
                                                  | A.TupleAccess(s, e) -> let i = expr builder e in build_tuple_access s (L.const_int i32_t 0) i builder true
                                                  | A.MatrixAccess (s, e1, e2) -> let i1 = expr builder e1 and i2 = expr builder e2 inbuild_matrix_access s (L.const_int i32_t 0) i1 i2 builder true
                                                  | A.PointerIncrement(s) -> build_pointer_increment s builder true
                                                  | A.Dereference(s) -> build_pointer_dereference s builder true
                                                  | _ -> raise (IllegalAssignment))
                                     and e2' = expr builder e2 in
                               ignore (L.build_store e2' e1' builder); e2'
              | A.Call ("print", [e]) | A.Call ("printb", [e]) ->
              L.build_call printf_func [| int_format_str ; (expr builder e) |]
```

```
                "printf" builder
              | A.Call ("printsl", [e]) | A.Call ("printbsl", [e]) ->
            L.build_call printf_func [| int_sl_format_str ; (expr builder e) |]
                "printf" builder
              | A.Call ("prints", [e]) -> let get_string = function A.StrLit s -> s | _ -
> "" in
              let s_ptr = L.build_global_stringptr ((get_string e) ^ "\n") ".str" builder in
            L.build_call printf_func [| s_ptr |] "printf" builder
              | A.Call ("printssl", [e]) -> let get_string = function A.StrLit s -> s | _ -
> "" in
              let s_ptr = L.build_global_stringptr (get_string e) ".str" builder in
            L.build_call printf_func [| s_ptr |] "printf" builder
              | A.Call ("printf", [e]) ->
            L.build_call printf_func [| float_format_str ; (expr builder e) |]
              "printf" builder
              | A.Call ("printfsl", [e]) ->
            L.build_call printf_func [| float_sl_format_str ; (expr builder e) |]
              "printf" builder
              | A.Call ("printc", [e]) ->
            L.build_call printf_func [| char_format_str ; (expr builder e) |]
              "printf" builder
              | A.Call ("printcsl", [e]) ->
            L.build_call printf_func [| char_sl_format_str ; (expr builder e) |]
              "printf" builder
              | A.Call (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.A.datatype with A.DataType(A.Void) -> ""
                                                       | _ -> f ^ "_result") in
                 L.build_call fdef (Array.of_list actuals) result builder
            in

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

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

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

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

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

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

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

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

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

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

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

            (* Add a return if the last block falls off the end *)
            add_terminal builder (match fdecl.A.datatype with
                A.DataType(A.Int) -> L.build_ret (L.const_int i32_t 0)
              | A.DataType(A.Float) -> L.build_ret (L.const_float float_t 0.0)
              | A.DataType(A.Char) -> L.build_ret (L.const_int i8_t 0)
              | A.DataType(A.Bool) -> L.build_ret (L.const_int i1_t 0)
              | A.DataType(A.Void) -> L.build_ret_void
              | A.TuplePointerType(t) -> (match t with
                                           A.Int -> L.build_ret (L.const_pointer_null (pointer_t i32_t))
                                         | A.Float -> L.build_ret (L.const_pointer_null (pointer_t float_t))
                                         | A.Char -> L.build_ret (L.const_pointer_null (pointer_t i8_t))
                                         | A.Bool -> L.build_ret (L.const_pointer_null (pointer_t i1_t))
                                         | _ -> raise (IllegalReturnType))
              | A.MatrixPointerType(t) -> (match t with
                                             A.Int -> L.build_ret (L.const_pointer_null (pointer_t i32_t))
                                           | A.Float -> L.build_ret (L.const_pointer_null (pointer_t float_t))
                                           | _ -> raise (IllegalReturnType))
              | A.MatrixTuplePointerType(t) -> (match t with
                                                  A.Int -
> L.build_ret (L.const_pointer_null (pointer_t i32_t))
                                                | A.Float -
> L.build_ret (L.const_pointer_null (pointer_t float_t))
                                                | _ -> raise (IllegalReturnType))

              | _ -> raise (IllegalReturnType)
            )
          in

          List.iter build_function_body functions;
          the_module
```

9.1.9 ml.ml

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

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 2 then
    List.assoc Sys.argv.(2) [ ("-a", Ast);  (* Print the AST only *)
                  ("-l", LLVM_IR);                (* Generate LLVM, don't check *)
                  ("-c", Compile) ]               (* Generate, check LLVM IR *)
  else Compile in
  let instring = Prep.process Sys.argv.(1) in
  let lexbuf = Lexing.from_string instring in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

9.1.10 Makefile

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

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

.PHONY : ml.native

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

# "make clean" removes all generated files

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

# "make pretty" generates pretty print ast generator

POBJS = parser.cmo scanner.cmo ast.cmo pretty.cmo

.PHONY : pretty
pretty : $(POBJS)
    ocamlc -o pretty $(POBJS)

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

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx ml.cmx prep.cmx

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

```

```
scanner.ml : scanner.mll
	ocamllex scanner.mll

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

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

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

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

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
ml.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo prep.cmo
ml.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx prep.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
prep.cmo:
prep.cmx:

# 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

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 ml.ml parser.mly README scanner.mll \
    semant.ml testall.sh $(TESTFILES:%=tests/%)

ml-llvm.tar.gz : $(TARFILES)
    cd .. && tar czf ml-llvm/ml-llvm.tar.gz \
        $(TARFILES:%=ml-llvm/%)
```

9.2 References

1. B.W. Kernighan and D.M. Ritchie. "Appendix A Reference Manual," in The C Programming Language, 2nd edition. Murray Hill, NJ: AT&T Bell Laboratories.
2. MicroC Programming Language http://www.cs.columbia.edu/~sedwards/classes/2016/4115-spring/microc-llvm.tar.gz
3. note-hashtag Final Report (final report formatting guidance) http://www.cs.columbia.edu/~sedwards/classes/2015/4115-fall/reports/note-hashtag.pdf