MATRX

Katie Pfleger, Julia Sheth, Alana Anderson,
Pearce Kieser, Nicholas Sparks

December 19, 2018

Contents

1 Introduction **7**
1.1 Motivation 7
1.2 Background 7

2 Language Tutorial **8**
2.1 Setup 8
2.1.1 Installation 8
2.1.2 Build the compiler 8
2.1.3 Add the compiler to $PATH 8
2.2 Basic Syntax 8
2.3 Example Programs 8
2.3.1 String Manipulation 8
2.3.2 Matrix Manipulation 9

3 Language Reference Manual **10**
3.1 Lexical Elements 10
3.1.1 Comments 10
3.1.2 Identifiers 10
3.1.3 Keywords 10
3.1.4 Constants 10
3.1.5 Operators 11
3.1.6 Separators 11
3.1.7 White Space 11
3.2 Data Types 12
3.2.1 Matrices 12
3.2.2 Strings 13
3.2.3 Integers 14
3.2.4 Floats 14
3.3 Chars 14
3.4 Expressions and Operators 15
3.4.1 Expressions 15
3.4.2 Assignment Operators 15
3.4.3 Arithmetic Operators 15
3.4.4 Comma Operator 15
3.4.5 Operator Precedence 15

3.5 Statements . . . 16
3.5.1 Expression Statement . . . 16
3.5.2 Conditional Statement . . . 16
3.5.3 While Statement . . . 16
3.5.4 For Statement . . . 17
3.5.5 Return Statement . . . 17
3.5.6 Null Statement . . . 17
3.6 Functions . . . 17
3.6.1 Function Declarations . . . 18
3.6.2 Function Definitions . . . 18
3.6.3 Function Calls . . . 18
3.6.4 The Main Function . . . 19
3.7 Scope . . . 19

4 Project Plan **20**
4.1 Process Used . . . 20
4.2 Style Guide . . . 21
4.3 Project Timeline . . . 21
4.4 Roles and Responsibilities . . . 21
4.5 Software Development Environment . . . 22
4.6 Project Log . . . 22

5 Architectural Design **23**
5.1 Architecture Diagram . . . 23
5.2 Scanner . . . 24
5.3 Parser . . . 24
5.4 Semantic Checking . . . 24
5.5 Code Generation . . . 24

6 Test Plan **25**
6.1 Test Programs . . . 25
6.1.1 Declaring/Manipulating Matrices . . . 25
6.1.2 Declaring/Manipulating Strings . . . 26
6.2 Test Suites . . . 29
6.2.1 Reasoning . . . 29
6.2.2 Automation . . . 30
6.3 Roles and Responsibilities . . . 30

7 Lessons Learned **31**
7.1 Katie Pfleger . . . 31
7.2 Julia Sheth . . . 31
7.3 Alana Anderson . . . 32
7.4 Pearce Kieser . . . 32
7.5 Nicholas Sparks . . . 32

8 Appendix **34**
8.1 Code Listing . . . 34
8.1.1 Git Log . . . 34
8.1.2 scanner.mll . . . 45
8.1.3 parser.mly . . . 47

8.1.4 ast.ml 48
8.1.5 sast.ml 50
8.1.6 semant.ml 52
8.1.7 codegen.ml 58
8.1.8 matrx.ml 66
8.1.9 matrix.c 67
8.1.10 Makefile 73
8.1.11 README 74
8.1.12 demo.mx 79
8.2 Tests 80
8.2.1 fail-assign1.err 80
8.2.2 fail-assign1.mx 80
8.2.3 fail-assign2.err 80
8.2.4 fail-assign2.mx 80
8.2.5 fail-assign3.err 81
8.2.6 fail-assign3.mx 81
8.2.7 fail-dead1.err 81
8.2.8 fail-dead1.mx 81
8.2.9 fail-dead2.err 81
8.2.10 fail-dead2.mx 81
8.2.11 fail-expr1.err 82
8.2.12 fail-expr1.mx 82
8.2.13 fail-expr2.err 82
8.2.14 fail-expr2.mx 82
8.2.15 fail-expr3.err 82
8.2.16 fail-expr3.mx 83
8.2.17 fail-float1.err 83
8.2.18 fail-float1.mx 83
8.2.19 fail-float2.err 83
8.2.20 fail-float2.mx 83
8.2.21 fail-for1.err 83
8.2.22 fail-for1.mx 84
8.2.23 fail-for2.err 84
8.2.24 fail-for2.mx 84
8.2.25 fail-for3.err 84
8.2.26 fail-for3.mx 84
8.2.27 fail-for4.err 84
8.2.28 fail-for4.mx 85
8.2.29 fail-for5.err 85
8.2.30 fail-for5.mx 85
8.2.31 fail-func1.err 85
8.2.32 fail-func1.mx 85
8.2.33 fail-func2.err 85
8.2.34 fail-func2.mx 86
8.2.35 fail-func3.err 86
8.2.36 fail-func3.mx 86
8.2.37 fail-func4.err 86
8.2.38 fail-func4.mx 86
8.2.39 fail-func5.err 86
8.2.40 fail-func5.mx 87

8.2.41 fail-func6.err 87
8.2.42 fail-func6.mx 87
8.2.43 fail-func7.err 87
8.2.44 fail-func7.mx 87
8.2.45 fail-func8.err 88
8.2.46 fail-func8.mx 88
8.2.47 fail-func9.err 88
8.2.48 fail-func9.mx 88
8.2.49 fail-global1.err 88
8.2.50 fail-global1.mx 88
8.2.51 fail-global2.err 89
8.2.52 fail-global2.mx 89
8.2.53 fail-if1.err 89
8.2.54 fail-if1.mx 89
8.2.55 fail-if2.err 89
8.2.56 fail-if2.mx 89
8.2.57 fail-if3.err 89
8.2.58 fail-if3.mx 90
8.2.59 fail-matrix-dims-3D.err 90
8.2.60 fail-matrix-dims-3D.mx 90
8.2.61 fail-matrix-dims.err 90
8.2.62 fail-matrix-dims.mx 90
8.2.63 fail-nomain.err 90
8.2.64 fail-printb.err 90
8.2.65 fail-printb.mx 91
8.2.66 fail-print.err 91
8.2.67 fail-print.mx 91
8.2.68 fail-return1.err 91
8.2.69 fail-return1.mx 91
8.2.70 fail-return2.err 91
8.2.71 fail-return2.mx 91
8.2.72 fail-while1.err 91
8.2.73 fail-while1.mx 92
8.2.74 fail-while2.err 92
8.2.75 fail-while2.mx 92
8.2.76 test-add1.mx 92
8.2.77 test-add1.out 92
8.2.78 test-arith1.mx 93
8.2.79 test-arith1.out 93
8.2.80 test-arith2.mx 93
8.2.81 test-arith2.out 93
8.2.82 test-arith3.mx 93
8.2.83 test-arith3.out 93
8.2.84 test-fib.mx 94
8.2.85 test-fib.out 94
8.2.86 test-float1.mx 94
8.2.87 test-float1.out 94
8.2.88 test-float2.mx 94
8.2.89 test-float2.out 95
8.2.90 test-float3.mx 95

8.2.91 test-float3.out 95
8.2.92 test-for1.mx 96
8.2.93 test-for1.out 96
8.2.94 test-for2.mx 96
8.2.95 test-for2.out 96
8.2.96 test-func1.mx 97
8.2.97 test-func1.out 97
8.2.98 test-func2.mx 97
8.2.99 test-func2.out 97
8.2.100 test-func3.mx 97
8.2.101 test-func3.out 98
8.2.102 test-func4.mx 98
8.2.103 test-func4.out 98
8.2.104 test-func5.mx 98
8.2.105 test-func6.mx 98
8.2.106 test-func6.out 99
8.2.107 test-func7.mx 99
8.2.108 test-func7.out 99
8.2.109 test-func8.mx 99
8.2.110 test-func8.out 99
8.2.111 test-func9.mx 100
8.2.112 test-func9.out 100
8.2.113 test-gcd2.mx 100
8.2.114 test-gcd2.out 100
8.2.115 test-gcd.mx 100
8.2.116 test-gcd.out 101
8.2.117 test-global1.mx 101
8.2.118 test-global1.out 101
8.2.119 test-global2.mx 101
8.2.120 test-global2.out 102
8.2.121 test-global3.mx 102
8.2.122 test-global3.out 102
8.2.123 test-hello.mx 102
8.2.124 test-hello.out 102
8.2.125 test-if1.mx 103
8.2.126 test-if1.out 103
8.2.127 test-if2.mx 103
8.2.128 test-if2.out 103
8.2.129 test-if3.mx 103
8.2.130 test-if3.out 103
8.2.131 test-if4.mx 103
8.2.132 test-if4.out 104
8.2.133 test-if5.mx 104
8.2.134 test-if5.out 104
8.2.135 test-if6.mx 104
8.2.136 test-if6.out 104
8.2.137 test-local1.mx 105
8.2.138 test-local1.out 105
8.2.139 test-local2.mx 105
8.2.140 test-local2.out 105

8.2.141 test-matadd.mx 105
8.2.142 test-matadd.out 105
8.2.143 test-matdet.mx 106
8.2.144 test-matdet.out 106
8.2.145 test-mat-dot2.mx 106
8.2.146 test-mat-dot2.out 106
8.2.147 test-mat-dot.mx 106
8.2.148 test-mat-dot.out 106
8.2.149 test-matmult.mx 106
8.2.150 test-matmult.out 107
8.2.151 test-mat-print2.mx 107
8.2.152 test-mat-print2.out 107
8.2.153 test-mat-print.mx 107
8.2.154 test-mat-print.out 107
8.2.155 test-matrix1.mx 107
8.2.156 test-matrix1.out 107
8.2.157 test-matscale.mx 108
8.2.158 test-matscale.out 108
8.2.159 test-mat-trans.mx 108
8.2.160 test-mat-trans.out 108
8.2.161 test-ops1.mx 108
8.2.162 test-ops1.out 109
8.2.163 test-ops2.mx 109
8.2.164 test-ops2.out 109
8.2.165 test-string1.mx 110
8.2.166 test-string1.out 110
8.2.167 test-string2.mx 110
8.2.168 test-string2.out 110
8.2.169 test-string-decl1.mx 110
8.2.170 test-string-decl1.out 111
8.2.171 test-string-decl2.mx 111
8.2.172 test-string-decl2.out 111
8.2.173 test-var1.mx 111
8.2.174 test-var1.out 111
8.2.175 test-var2.mx 111
8.2.176 test-var2.out 112
8.2.177 test-while1.mx 112
8.2.178 test-while1.out 112
8.2.179 test-while2.mx 112
8.2.180 test-while2.out 112

1 Introduction

1.1 Motivation

Machine learning (ML) is the area of computational science that focuses on analyzing and interpreting patterns and structures in data in order to enable learning, reasoning, and decision making. While the use of machine learning has been around since the turn of the century, it has only recently become mainstream in the industry. Today, 51% of enterprises across a variety of industries are deploying machine learning in production.[1] In fact, job titles such as "machine learning engineer," "deep learning engineer," and "data scientist" are already widely used terms. As these engineers will tell you, though, machine learning really boils down to one thing: matrices.

A matrix is a two-dimensional array of scalars with one or more columns and one or more rows. Matrix manipulations are often essential to machine learning algorithms, where they are used as the input data when training algorithms. However, implementing these operations in common programming languages (such as C, C++, or python) can be extremely complicated and time-consuming. While libraries and tools with more robust matrix manipulation tools exist, they are often expensive and syntactically complex. With this motivation, we have decided to build a simple language that supports matrix operations by design.

1.2 Background

The MATRX language is a general purpose programming language that is designed to support matrix manipulations out-of-the-box. The syntax and semantics of our language closely resemble that of the C programming language. With the MATRX language, the user can declare matrices using a simple and intuitive notation. Further, she can perform powerful computations such as multiplication, and transpose with a single function.

In addition to the matrix type, our language supports a primitive string type. Similar to the matrix type, the string can easily be manipulated using a set of built in functions such as length, substring, and concatenate.

[1] Lorica, B., Nathan, P. (2018). The State of Machine Learning Adoption in the Enterprise. O'Reilly Media, Inc.

2 Language Tutorial

2.1 Setup

To setup the environment our language requires a few dependencies are installed. This section will describe the installation process of the matrx compiler on Ubuntu 16.04.

2.1.1 Installation

The following commands will install the required dependencies:

```
sudo apt install ocaml llvm llvm-runtime m4 build-essential autoconf automake opam clang
opam init
opam install llvm.3.8
eval `opam config env`
```

The above commands will install all the required dependencies.

2.1.2 Build the compiler

To build the compiler, unpack the submitter matrx.tar.gz file and navigate to that directory. Run `make`. This will build the compiler and begin running the test suite. Running `make all` will only build the compiler, `./testall.sh` will test the compiler against our test suite.

2.1.3 Add the compiler to $PATH

To use the compiler outside of this directory you should move the `matrix.native` file into a meaningful directory line `/usr/bin` and insure that directory is a part of your `$PATH` so you can run `matrx.native fileToCompile > output.s` anywhere on your computer.

2.2 Basic Syntax

Our languages uses a simple C-like syntax and program structure, where executables run from their main function and all functions must have a return statement. In addition to the basic types supported by C, MATRX supports both strings and matrices as primitive types.

2.3 Example Programs

Below we give two example programs in our language, which demonstrate its string and matrix manipulation capabilities.

2.3.1 String Manipulation

MATRX supports easy and intuitive string operations, allowing the users to declare, print, and manipulate strings. As with all types in our language, a string must be declared before it is assigned. Once declared, a string may be passed into a number of out-of-the-box functions including `length`, `get_char`, `sequals`, `sconcat`, and `substring`. In the example below, we demonstrate the use of each of these functions.

```
int main()
{
  string s1;
  s1 = "This is an example ";
  printstr(s1);
  printstr(get_char(s1, 0));

  printb(sequals("MATRX", "MATRX"));
  printstr(sconcat("This is an example ", "of a string in MATRX!"));
  printstr(substring("This is an example of a string in MATRX!", 3, 7));
  return 0;
}
```

2.3.2 Matrix Manipulation

MATRX supports the matrix data type, which allows for easy declaration and manipulation of two dimensional int arrays. In order to declare a matrix, the identifier must be specified, demonstrated in the example below.

```
int main()
{
    matrix m1; /* declares a matrix */
    matrix m2 = [[1, 2], [3, 4]]; /* declares a matrix */
                                  /*      [1, 2]       */
                                  /*      [3, 4]       */
}
```

MATRX supports built-in methods for the matrix data type, such as print, dot product, transpose, addition, determinant, matrix multiplication, and scalar multiplication.

3 Language Reference Manual

3.1 Lexical Elements

This chapter describes the lexical elements that make up MATRX source code after processing. We refer to these elements as tokens. We specify five types of tokens: keywords, identifiers, constants, operators, and separators.

3.1.1 Comments

The characters `/* introduce a comment, which terminates with the characters */`.

3.1.2 Identifiers

Identifiers are sequences of characters used for naming variables and functions. Users may use letters and the underscore character _ in identifiers. Identifiers are case sensitive, such that `foo` and `FOO` are two different identifiers.

3.1.3 Keywords

Keywords are special identifiers reserved for use as part of the programming language itself. In MATRX, we have the following keywords:

```
if, else, for, while, return, int, bool, float, string, void, matrix
```

3.1.4 Constants

Integer Constants: An integer constant is a sequence of digits.

```
/* example integer constants */
2018
42
1
```

Real Number Constants: A real number constant is a value that represents a fractional number. It consists of a sequence of digits which represent the integer, a decimal point, and a sequence of digits which represent the fraction.

```
/* example real number constants */
4.2
4.
4
.42
```

String Constants: A string constant is a sequence of zero or more characters, digits, and escape sequences enclosed within double quotation marks. All string constants contain a null termination character () as their last character to indicate the end of the string.

```
/* a simple string constant */
"matrix languages are the best languages"
```

3.1.5 Operators

An operator is a special token that performs an operation. Full coverage of operators can be found in Chapter 3 of this Language Reference Manual.

3.1.6 Separators

A separator separates tokens. White space is a separator, but not a token. We have the following separators:

```
( ) [ ] { } ; , .
```

3.1.7 White Space

White space is the collective term used for several characters: the space character, the tab character, the newline character, the vertical tab character, and the form-feed character. White space is ignored (outside of string and character constants), and is therefore optional, except when it is used to separate tokens.

This means that:

```
#include <stdio.h>
int main()
{
  printf( "hello, world\n" );
  return 0;
}
```

is functionally the same as:

```
#include int main() { printf( "hello, world\n" ); return 0; }
```

White space is not required between operators and operands, nor is it required between other separators and that which they separate. This means that:

```
matrix m = [ [0, 1]
              [2, 3] ];
```

is equivalent to:

```
matrix m = [[0, 1][2, 3]];
```

In string constants, spaces and tabs are included in the string. This means that:

```
"This is a string with spaces."
```

Is not the same as:

```
"Thisisastringwithspaces."
```

3.2 Data Types

3.2.1 Matrices

A matrix is a data structure that lets you store a two dimensional array of numbers. A matrix has at least one row and at least one column.

Declaring Matrices: Matrices can be declared by specifying the identifier. Matrices can only hold data that is of type integer. Note that you can declare a matrix without initializing it (see section 3.2.1.2 for information on how to initialize a matrix).
Here is an example:

```
matrix m; /* declares a matrix*/
```

Initializing Matrices: You can initialize elements in a matrix when you declare it by listing each row as a list of elements separated by commas and enclosed by square braces. The data type contained by a matrix and the number of rows and cols is determined when it is initialized. Note that white space does not change the initialization (see 1.7). Here is an example:

```
matrix a = [ [1, 2] [3, 4] ]; /* declares a matrix */
                               /*     [1, 2]        */
                               /*     [3 4]         */

matrix b = [ [1, 2]
    [3, 4] ]; /* declares the same matrix as above */
```

When a matrix is declared with an incompatible number of rows and cols, we throw an error:

```
matrix m;
m = [ [1]
      [3, 4] ]; /* this will throw an error */
m = [ [1]
      [3, 4]
      [5, 6]; /* this will throw an error */
```

Printing Matrices: We provide the built-in function printm(matrix m) to print a matrix.
Here is an example:

```
matrix a = [ [1, 2] [3, 4] ]; /* declares a matrix */
                               /*     [1, 2]        */
                               /*     [3 4]         */
printm(a); /* prints the matrix a as [2 2] */
           /*                        [3 4] */
```

Matrix Arithmetic Methods: We provide built-in functions for mathematical operations on matrices, including determinant, dot product, transpose, matrix addition, and matrix multiplication. Here are the exact methods that we provide:

- `transpose(matrix m)`: returns the matrix transposed

- `matmult(matrix m, matrix n)`: returns the resulting matrix from multiplying matrices m and n
- `matadd(matrix m, matrix n)`: returns the resulting matrix from adding matrices m and n
- `dot(matrix m, matrix n)`: returns the dot product of matrices m and n
- `det(matrix m, int dimension)`: returns the determinant of matrix m (which must be a square matrix dimensions matching the "dimension" argument

3.2.2 Strings

The string data type is a string constant made up of characters. All string constants contain a null termination character () as their last character to indicate the end of the string.

String Declaration: A string can be declared by specifying an identifier and then, between quotation marks, the list of characters that string is supposed to hold.
Here is an example:

```
string s;
s = "hello world";
```

String Methods: We provide a built-in function for printing strings, printstr. It accepts a string as it's only argument and prints the string to standard output. Here is an example:

```
string s;
s = \test";
printstr(s); /* prints test */
```

We also provide built-in functions for getting information about strings, including length, the character at a given index, the substring between given indices, and whether or not one string equals another:

- `length()`: returns the length of the string
- `get_char(string s, int index)`: returns the character at the given index
- `sequals(string s1, string s2)`: returns true when the two strings are the same, otherwise returns false
- `substring(string s, int start_index, int end_index)`: returns a string which only contains the characters from s that are between the start_index and the end_index

Here are some examples:

```
string s1 = \hello";
string s2 = \world";
int i;

i = length(s1);
print(i); /* prints 5 */

printb(sequals(s1, s2)); /* prints 0 */
```

We provide a built-in function to manipulate strings by concatenating two strings. The syntax is as follows:

- `sconcat(string s1, string s2)`: returns a string in which s2 is attached to the end of s1

Here is an example:

```
string s1 = \hello";
string s2 = \world";

string s3 = sconcat(s1, s2);
printstr(s3); /* prints \hello world" */
```

3.2.3 Integers

Integer types can be used for storing whole number values. We support a 32-bit int data type, which can hold integer values in the range of 2,147,483,648 to 2,147,483,647.

Here are some examples of declaring and defining integer variables:

```
int a;
int a = 10;
```

3.2.4 Floats

The float data type's minimum value is stored in the FLT_MIN, and should be no greater than 1e-37. Its maximum value is stored in FLT_MAX, and should be no less than 1e37.

```
float f;
float f = 10.0;
```

3.3 Chars

The char data type allows for storing a single character.

```
char c;
char c = 'a';
```

3.4 Expressions and Operators

3.4.1 Expressions

An expression consists of at least one operand and zero or more operators. An operand is defined as a typed object such as a constant, variable, or function call that returns a value. An operator specifies an operation to be performed on the operand(s). Here are some examples:

```
42
2+2
```

We let parentheses group subexpressions. Innermost expressions are evaluated first. In the example below, (3 + 10) is evaluated to 13 and (2 * 6) is evaluated to 12. Then, 12 is subtracted from 13. Finally, the result of that subtraction, 1, is multiplied by 2.

```
(2 * ((3 + 10) - (2 * 6)))
```

3.4.2 Assignment Operators

Assignment operators store values in variables. The standard operator = stores the value of its right operand in the variable specified by its left operand. The left operand cannot be a literal or constant.

Here are some examples:

```
int x = 10;
float y = 41.1 + 0.9;
```

3.4.3 Arithmetic Operators

We provide operators for standard arithmetic operations: addition, subtraction, multiplication, and division, along with modular division and negation.

Here are some examples:

```
a = 5 + 3;
b = 43.5 - 1.5;
c = 5 * 10;
```

3.4.4 Comma Operator

A pair of expressions separated by a comma is evaluated left-to-right and the value of the left expression is discarded. The type and value of the result are the type and value of the right operand. This operator groups left-to-right. It should be avoided in situations where comma is given a special meaning, for example in actual arguments to function calls and lists of initializers.

3.4.5 Operator Precedence

The following is a list of types of expressions, presented in order of highest precedence first. Sometimes two or more operators have equal precedence; all those operators are applied from left to right unless stated otherwise.

1. function calls

2. unary operators (including logical negation, increment, decrement, unary positive, unary negative, indirection operator, address operator, type casting, and sizeof expressions)
3. multiplication, division, and modular division expressions (including matrix operations of these types)
4. addition and subtraction expressions (including matrix operations of these types)
5. greater-than, less-than, greater-than-or-equal-to, and less-than-or-equal-to expressions
6. equal-to and not-equal-to expressions
7. conditional expressions
8. all assignment expressions, including compound assignment
9. comma operator expressions

3.5 Statements

Except as indicated, statements are executed in sequence.

3.5.1 Expression Statement

Most statements are expression statements, of the form:

```
expression;
```

Usually expression statements are assignments or function calls.

3.5.2 Conditional Statement

The two forms of the conditional statement are:

```
if ( expression ) statement
if ( expression ) statement else statement
```

In both cases the expression is evaluated and if it is non-zero, the first substatement is executed. In the second case the second substatement is executed if the expression is 0. As usual the `else` ambiguity is resolved by connecting an else with the last encountered else-less if.

3.5.3 While Statement

The while statement has the form:

```
while ( expression ) statement
```

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

3.5.4 For Statement

The for statement has the form:

```
for ( expression-1opt ; expression-2opt ; expression-3opt ) statement
```

This statement is equivalent to:

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

Thus 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 becomes 0; the third expression typically specifies an incrementation which is performed after each iteration. Any or all of the expressions may be dropped. A missing expression-2 makes the implied while clause equivalent to "while(1)"; other missing expressions are simply dropped from the expansion above.

3.5.5 Return Statement

A function returns to its caller by means of the return statement, which has one of the forms:

```
return ;
return ( expression ) ;
```

In the first case no value is returned. In the second case, the value of the expression is returned to the caller of the function. If required, the expression is converted, as if by assignment, to the type of the function in which it appears. Flowing off the end of a function is equivalent to a return with no returned value.

3.5.6 Null Statement

The null statement has the form

```
;
```

A null statement is useful to carry a label just before the } of a compound statement or to supply a null body to a looping statement. In the following example, a null statement is used as the body of the loop:

```
for (i = 1; i*i < n; i++)
    ;
```

3.6 Functions

We allow users to define functions to separate parts of a program into distinct subroutines. To write a function, you must create a function definition. Every program requires at least one function, the main function, where the program's execution begins (see 3.5.6).

3.6.1 Function Declarations

You write a function declaration to specify the name of a function, a list of parameters, and the function's return type. A function declaration ends with a semicolon. You should write the function declaration above the first use of the function. Function declarations have the form:

```
return-type function-name (parameter-list);
```

The return type indicates the data type of the value returned by the function. A function that does not return any data type has the return type void. The function name can be any valid identifier. The parameter list consists of zero or more parameters, separated by commas. A single parameter consists of a data type and an identifier.

Here is an example:

```
int add(int a, int b);
```

3.6.2 Function Definitions

A function definition specifies what the function does. Function definitions must specify the name of a function, the list of parameters, the return type, and the body of the function. Function definitions have the form:

```
return-type function-name (parameter-list)
{
  function-body
}
```

Here is an example:

```
int add(int a, int b)
{
  return a + b;
}
```

3.6.3 Function Calls

Functions are called by using its name and supplying the necessary parameters. Function calls have the form:

```
function-name (parameters)
```

A function call can make up an entire statement or be used as a subexpression:

```
/* as an entire statement */
sconcat(\hello", \world");

/* as a subexpression */
string s;
s = sconcat (\hello", \world");
```

In the example above, even though the parameter `a` is modified in the function `foo`, the variable `x` that is passed to the function does not change when `foo (x)` is called. The original value of x is only changed when we reassign `x = foo (x)`.

3.6.4 The Main Function

Every program requires at least one function, called `main`. This is where the program begins executing. The main function does not need a declaration, but must be defined.

The return type for main is always `int`. You do not have to specify the return type for main; however, you cannot specify that it has a return type other than `int`. In general, the return value from main indicates the program's exit status. A value of zero or EXIT SUCCESS indicates success and EXIT FAILURE indicates an error. Otherwise, the significance of the value returned is implementation-defined.

The 'main' function can be written to accept no parameters or to accept parameters from the command line. To accept parameters from the command line, the function must have two parameters: argc (an int specifying the number of command line arguments) and argv (a one-dimensional matrix of parameters).

Here are some examples:

```
/* main function with no arguments */
int main ()
{
  printstr ("Hello World!");
  return 0;
}
```

3.7 Scope

Scope refers to what parts of the program can "see" a declared object. A declared object can be visible only within a particular function, or within a particular file, or may be visible to an entire set of files by way of including header files and using extern declarations. Unless explicitly stated otherwise, declarations made at the top-level of a file (i.e., not within a function) are visible to the entire file, including from within functions, but are not visible outside of the file. Declarations made within functions are visible only within those functions. A declaration is not visible to declarations that came before it.
Here are some examples:

```
int x;
x = 5;
int y = x + 10; /* this will work because x is already defined */

int x = y + 10; /* this will not work as y has not been defined */
int y = 5;
```

4 Project Plan

4.1 Process Used

Planning Process: Throughout the semester, we met twice a week: once as a group and once with our TA. Our weekly group meetings took the form of a stand-up, in which we would discuss what each of us had been working on the previous week and what we were planning to work on next. Prior to these meetings, our managers would set the plan for the next couple of milestones and communicate them to the group.

Specification Process: Early in the semester, we met to discuss our language and some the features we wanted to implement. We knew we wanted to create a language that could support matrix manipulations such as multiplication, dot product, etc. It wasn't until our language reference manual, however, that these ideas became concrete specifications. In order to do so, we had to consider tradeoffs about the usability of our language and the ease of building it. For example, we considered questions such as "should we support matrices of multi-dimensions" and "should we allow non-numeric types to be stored in matrices".

Development Process: Our development process followed the stages of compilation. For our first deliverable, we implemented the scanner and parser for our language. Next, we started working on the AST, SAST, and semant. Finally, we integrated the codegen to be able to run our language as a whole. Once we had codegen working, we started to add more functionality by integrating a C library for matrix manipulations. As mentioned, we met once a week with our TA, Dean Deng. During these meetings, we would check in to see if we were approaching things the right way and go over any implementation questions we had. One practice we added near the end of our development process was weekly demos where one member of the group would walk through a feature they had recently implemented. We found this a great way to motivate team members toward specific goals as well as keep the group up to date on additions to the codebase.

Testing Process: See section 6 for our testing process in detail.

4.2 Style Guide

Git Requirements:

- We required that the code must compile prior to pushing new changes.
- We required that each developer runs 'git pull' before pushing in order to merge previous changes.
- We required that each developer includes a detailed commit message so that the rest of the team would be able to follow the commit history.
- We added several generated files and targets to our .gitignore file in order to keep the repository clean. These files included parser generated files such as parser.mli and parser.ml, ocamlbuild targets such as files with .byte and .native extensions, and other unnecessary generated files with certain extensions (i.e., .o, .a, .cmi, etc.).

Comment Requirements:

- We required that our team members utilize comments in the same way in order to keep our code style consistent. Every comment added to our code precedes the code that it references.

Indentation Requirements:

- We required that our team members use consistent indentation throughout our files. We generally used one tab for indentation so that spacing would be aligned and the code would be easy to parse.

Naming Requirements:

- We followed MicroC's naming conventions for filenames.

4.3 Project Timeline

Date	Milestone
September 19	language proposal and white paper complete
October 16	language reference manual complete
October 16	scanner and parser complete
November 15	semantics and type-checking complete
November 15	"hello world" runs (without matrix functionality)
December 5	code generation complete
December 9	"hello world" runs (with matrix functionality)
December 15	regression testing complete
December 17	final report and presentation complete

4.4 Roles and Responsibilities

Katie Pfleger (Co-Manager): LRM, project planning/coordination, code generation (string implementation)

Julia Sheth (Co-Manager): LRM, project planning/coordination, C libraries, slide deck

Alana Anderson (Language Guru): LRM, white paper, code generation (string implementation)

Pearce Kieser (Software Architect): scanner, parser, code generation (matrix implementation), git setup

Nicholas Sparks (Tester): scanner, parser, code generation (matrix implementation), test suite

4.5 Software Development Environment

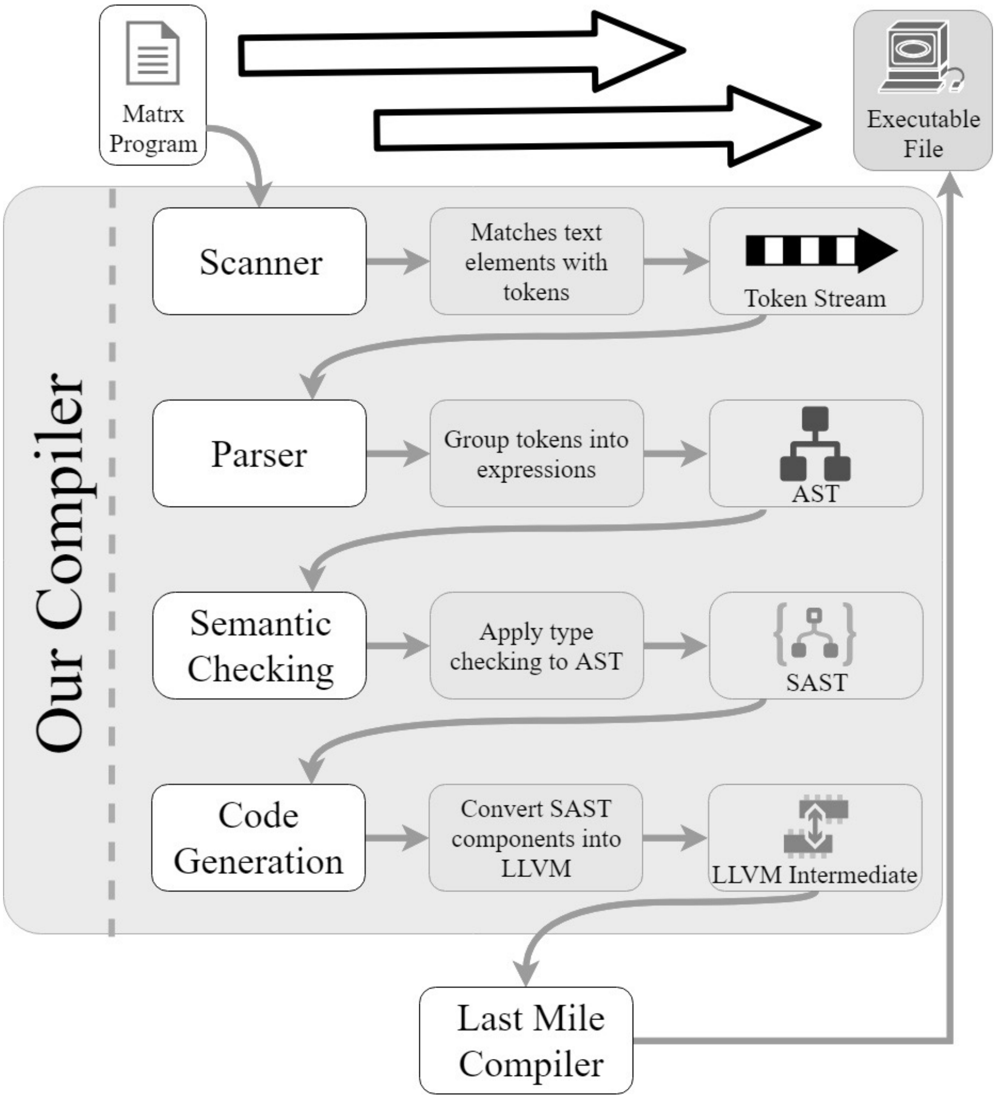
Programming Language: We implemented our compiler using Ocaml (version 4.00.1), with Ocamlyacc and Ocamllex extensions for compiling the scanner and parser front end. We implemented the matrix library in C.
Development Environments: Different members preferred to code in different environments including: vim, Visual Studio, and Sublime Text.

4.6 Project Log

See appendix section 8.2 for our project log.

5 Architectural Design

5.1 Architecture Diagram



5.2 Scanner

Implementation by Pearce and Nick

The scanner is responsible for taking in the input of a program and generating the tokens which will be read in the parser. During this phase, all of the white spaces are taken out and tokens are generated for anything that has syntactic meaning in the language. This includes all of the variable names, any braces or brackets as well as the string, integer, and float literals.

5.3 Parser

Implementation by Pearce and Nick

The parser is responsible for using the stream of tokens out of the scanner to construct an abstract syntax tree. Our implementation closely follows that of the MicroC compiler. The program is a series of declarations which can be variable declarations (globals) or function declarations.

5.4 Semantic Checking

Matrix implementation by Pearce and Nick
String implementation by Alana and Katie

The semantic checker performs a depth-first walk over the AST that was generated by the parser, verifying the semantics of the program. It contains a table of variable names and functions (symbol table), which are used to check that the program does not use any undeclared variables or functions. It also includes our built-in functions for strings and matrices.

5.5 Code Generation

Implementation by Pearce and Nick
String implementation by Alana and Katie

The code generator takes the semantically checked AST and returns an LLVM module. It is also responsible for linking to our C library, which includes our implementation for string and matrix manipulation.

6 Test Plan

6.1 Test Programs

Below we give several representative MATRX programs, along with the expected/actual output of the programs their generated LLVM code. We demonstrate simple unit tests, failure tests, and larger "black box" tests.

6.1.1 Declaring/Manipulating Matrices

This is a simple unit test program, which is designed to test declaring and printing a matrix in our language.

```
int main()
{
    matrix a = [[1, 2] [3, 4]];
    printm(a);
    return 0;
}
```

Below is the expected/actual output of this program.

```
 1 2
 3 4
```

Below is the generated LLVM code for this program.

```
; ModuleID = 'Matrx'
source_filename = "Matrx"

%struct.matrix = type { i32, i32, i32**, i32 }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

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

declare i32 @display(%struct.matrix*)

declare %struct.matrix* @initMatrix_CG(i32, i32)

declare %struct.matrix* @storeVal(%struct.matrix*, i32)

declare %struct.matrix* @transpose(%struct.matrix*)

declare %struct.matrix* @inverse(%struct.matrix*)

declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @dotProduct(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @timesScalar(%struct.matrix*, i32)

declare %struct.matrix* @determinant(%struct.matrix*, i32)
```

```
declare i8* @string_get(i8*, i32)

declare i32 @string_length(i8*)

declare i8* @string_concat(i8*, i8*)

declare i32 @string_equals(i8*, i8*)

declare i8* @string_substr(i8*, i32, i32)

define i32 @main() {
entry:
  %a = alloca %struct.matrix*
  %matrix_init = call %struct.matrix* @initMatrix_CG(i32 2, i32 2)
  %store_val = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, i32 1)
  %store_val1 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, i32 2)
  %store_val2 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, i32 3)
  %store_val3 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, i32 4)
  store %struct.matrix* %matrix_init, %struct.matrix** %a
  %a4 = load %struct.matrix*, %struct.matrix** %a
  %printm = call i32 @display(%struct.matrix* %a4)
  ret i32 0
}
```

We also include test cases to catch failures in our language. Below is a program that declares a matrix with invalid dimensions in order to ensure that an error is thrown.

```
int main(){
matrix a;
a = [[1,2][3]];
}
```

Below is the expected/actual output of this program.

```
Fatal error: exception Failure("Invalid matrix dimensions")
```

6.1.2 Declaring/Manipulating Strings

As with matrices, we will show a simple unit test for our string implementation. Below is a small program to test the `get_char()` method, which gets the character at a specific index.

```
int main()
{
    string s1 = "This is an example";
    printstr(get_char(s1, 0));
    return 0;
}
```

Below is the expected/actual output of this program.

```
T
```

Below is the generated LLVM code for this program.

```
; ModuleID = 'Matrx'
source_filename = "Matrx"

%struct.matrix = type { i32, i32, i32**, i32 }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@tmp = private unnamed_addr constant [19 x i8] c"This is an example\00"

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

declare i32 @display(%struct.matrix*)

declare %struct.matrix* @initMatrix_CG(i32, i32)

declare %struct.matrix* @storeVal(%struct.matrix*, i32)

declare %struct.matrix* @transpose(%struct.matrix*)

declare %struct.matrix* @inverse(%struct.matrix*)

declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @dotProduct(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @timesScalar(%struct.matrix*, i32)

declare %struct.matrix* @determinant(%struct.matrix*, i32)

declare i8* @string_get(i8*, i32)

declare i32 @string_length(i8*)

declare i8* @string_concat(i8*, i8*)

declare i32 @string_equals(i8*, i8*)

declare i8* @string_substr(i8*, i32, i32)

define i32 @main() {
entry:
  %s1 = alloca i8*
  store i8* getelementptr inbounds ([19 x i8], [19 x i8]* @tmp, i32 0, i32 0), i8** %s1
  %s11 = load i8*, i8** %s1
  %string_get = call i8* @string_get(i8* %s11, i32 0)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* %string_get)
  ret i32 0
}
```

Lastly, we will show a "black box" test, which is designed to test multiple features in a larger program. This program demonstrates the `sequals`, sconcat, and `substring` methods for strings in our language.

```
int main()
{
    printb(sequals("MATRX", "MATRX"));
    printstr(sconcat("This is an example ", "of a string in MATRX!"));
    printstr(substring("This is an example of a string in MATRX!", 3, 7));
    return 0;
}
```

Below is the expected/actual output of this program.

```
1
This is an example of a string in MATRX!
s is
```

Below is the generated LLVM code for this program.

```
; ModuleID = 'Matrx'
source_filename = "Matrx"

%struct.matrix = type { i32, i32, i32**, i32 }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@tmp = private unnamed_addr constant [6 x i8] c"MATRX\00"
@tmp.3 = private unnamed_addr constant [6 x i8] c"MATRX\00"
@tmp.4 = private unnamed_addr constant [22 x i8] c"of a string in MATRX!\00"
@tmp.5 = private unnamed_addr constant [20 x i8] c"This is an example \00"
@tmp.6 = private unnamed_addr constant [41 x i8] c"This is an example of a string in MATRX!\00"

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

declare i32 @display(%struct.matrix*)

declare %struct.matrix* @initMatrix_CG(i32, i32)

declare %struct.matrix* @storeVal(%struct.matrix*, i32)

declare %struct.matrix* @transpose(%struct.matrix*)

declare %struct.matrix* @inverse(%struct.matrix*)

declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @dotProduct(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @timesScalar(%struct.matrix*, i32)

declare %struct.matrix* @determinant(%struct.matrix*, i32)

declare i8* @string_get(i8*, i32)

declare i32 @string_length(i8*)

```

```
declare i8* @string_concat(i8*, i8*)

declare i32 @string_equals(i8*, i8*)

declare i8* @string_substr(i8*, i32, i32)

define i32 @main() {
entry:
  %string_equals = call i32 @string_equals(i8* getelementptr inbounds ([6 x i8], [6 x i8]* @tmp.3, i32 0, i32 0), i8* getelementptr inbounds ([6 x i8], [6 x i8]* @tmp, i32 0, i32 0))
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %string_equals)
  %string_concat = call i8* @string_concat(i8* getelementptr inbounds ([20 x i8], [20 x i8]* @tmp.5, i32 0, i32 0), i8* getelementptr inbounds ([22 x i8], [22 x i8]* @tmp.4, i32 0, i32 0))
  %printf1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* %string_concat)
  %string_substr = call i8* @string_substr(i8* getelementptr inbounds ([41 x i8], [41 x i8]* @tmp.6, i32 0, i32 0), i32 3, i32 7)
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* %string_substr)
  ret i32 0
}
```

6.2 Test Suites

Our test suite resides in the `/tests` directory. We have three possible extensions for our test files: `.mx` for MATRX programs, `.out` for expected output, and `.err` for expected errors.

6.2.1 Reasoning

As demonstrated in the previous section, our test suite was designed to resemble that of the MicroC compiler. Specifically, we wrote a minimum of two tests for each feature: one intended to pass and one intended to fail. We also included many smaller tests in order to verify smaller features such as simple operators, function calls, and variable declarations. Lastly, we included several "black box" tests (as demonstrated above), which test multiple components at once.

Before the project was able to compile, much of the testing was done through the OCaml interpreter. This helped us to get the functions needed for the project to compiler debugged, as OCaml does no favors in helping one understand small details of the language. Primarily, we copied and pasted the Ast and Sast structures, created sample values, and fed those into functions to see the results. Without doing this, tracking down the root cause of errors was very difficult. By the time we were able to compile, we had a deeper understanding of how the compiler and OCaml worked, and the errors were easier to understand and fix. At that point, simple test cases were all that were needed to move forward.

6.2.2 Automation

To automate our process, we created a compile.sh file to print out the Ast, Sast, LLVM, then compile, and execute. This showed us everything we needed to know in a single step. We also used a testall.sh file, similar to that of MicroC, to run regression testing. This was used to ensure that the build was stable before committing new code.

6.3 Roles and Responsibilities

As the tester, Nick took on the role of setting up our test suite. Each team member wrote test cases for his/her own features as needed.

7 Lessons Learned

7.1 Katie Pfleger

What I learned: As co-manager, I learned that weekly updates, planning, and communication are essential to effectively working as a team. Our team met twice each week, once for a standup and once with our TA, Dean. These meetings allowed us to communicate in person issues we were dealing with and let us come up with a game plan to deal with roadblocks. At times we struggled with communication, but having these meetings held each of us accountable for contributing to the project and we were able to resolve technical issues as a group.

My advice: My advice is to start early and create weekly goals that are feasible to deliver on. For our hello world deliverable, we tried to implement both string and matrix data types, but quickly discovered that these features would take much longer to complete. Had we come up with small, incremental deliverables such as first implementing string declarations, then string manipulations, then matrix declarations, and so on, I believe our team would have found less issues with communication and feature implementation. I would also advise to be transparent with your TA about issues you were dealing with. Dean was always so helpful with any issue we had. Even the smallest issue such as OCaml syntax can become a roadblock, so I would advise to deal with it early and openly.

7.2 Julia Sheth

What I learned: Before this semester, I had never worked on a semester-long group CS project. Being one of the co-managers of this team, I quickly learned just how important communication, planning, and flexibility are to creating a successful work environment. It is important for everyone to know what our group's goals are and to communicate what they are working on, but it is also important to be prepared for things to not always go according to plan. Weekly team stand-ups and meetings with our TA Dean were essential to keeping us on track and to keeping everyone up-to-date on our individual progress. Another big lesson that I took away from this project was that creating achievable, incremental goals makes sticking to a plan much easier. At the beginning, when we were first trying to get Hello Worlds working, we tried to get all of our functionality working at once with our individual efforts all very divided. However, we soon realized it was much better to focus on implementing one small feature well before moving on than trying to implement several features all at the same time. Finally, I learned how much time and care goes into every decision made about a programming language, no matter how big or small.

My advice: My advice would be to define clear expectations for communication within the group from the beginning so everyone knows how to keep themselves and others updated and to establish a weekly meeting time for group stand-ups. Additionally, try to get the Hello World working much earlier than the deadline so that you have time to first implement with MVP requirements and then build up and add features from there. Overall, never be afraid to ask questions whether it is to the TA or to other members of your group. Everyone is just starting out using OCaml and it is helpful for everyone to work through those questions together.

7.3 Alana Anderson

What I learned: This was my first real experience working with a group on a development project. Consequently, I had a lot of learning to do when it came to programming style, git basics, and so on. As the language guru, I was responsible for making a lot of difficult decisions and explaining their trade-offs to the group. I think this is a good skill that will always be useful in the industry as a software developer or otherwise. Lastly, I learned that communication is something that is easy to say and hard to do. There were several times where our group struggled to communicate about what we were working on or where we were stuck. The only way we were able to push through was by being honest with ourselves and each other.

My advice: My advice is to be totally transparent with your group. If you don't understand how something works, don't be afraid to ask. OCaml is a tricky language to figure out and even harder when you are building upon someone else's code. I would also encourage students to meet regularly with their team and mentor. Our mentor was incredibly helpful in helping us identify edge cases in our language and work well together as a team.

7.4 Pearce Kieser

What I learned: This was a difficult project. I was most surprised by the level of care necessary in coding a compiler. Each step of the compiler's development we were reminded of ambiguities and problems that had come up earlier in the design of our language. The decisions from the beginning of this project need to be well thought out. None of the compiler itself was that complicated to implement. We didn't have to use many fancy algorithms. The most interesting part of the development was the AST and SAST. These smart data structures really helped ease implementation difficulty.

My advice: Start coding early. Choose to implement features that build on each other. Don't be too ambitious right away. Our group struggled with our hello world submission because we tried to implement matrices at the same time that we were required to add strings to the language for the hello world project. Had we done our work in the order (static strings -¿ mutable strings -¿ matrices) I think these smaller easier goals would have let everyone get more familiar with the architecture of the compiler and lead to more effective team.

7.5 Nicholas Sparks

What I learned: Communication and planning are very important, and ideas don't always pan out. I've come to love and hate OCaml. On the love side, the code is so short and simple. On the hate side, I felt like I was a monkey bashing a keyboard until it worked. OCaml shows that you do not need verbose code to write effective and efficient code. Additionally, just because all of the individual components compile (Parser, Ast, Sast, Semant, Codegen), that does not mean they will work together as expected. Initially the parser seemed perfectly fine, but I would have to frequently revisit and revise the Parser and work my way all the way up to Codegen.

My advice: Expect things not to go as planned. Work on getting code to compile as soon as possible. This was the major roadblock in the entire project. Without a project that would compile with our features, we were unable to test, debug, add new features, etc. We were stuck until we were able to compile. Once that hurdle was cleared, everything came together and it became much easier to implement features.

8 Appendix

8.1 Code Listing

8.1.1 Git Log

```
commit 316d083e7498d1c9c1f1fd2b981f5f6fcf89b077
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Dec 18 22:03:21 2018 -0500

    removed inverse function

commit 3536ec24e958051860ff62308b395908275832c0
Author: Nicholas Sparks <>
Date:   Mon Dec 17 15:59:27 2018 -0500

    demo update

commit b09442cd2c76b8da9b4fd6d13f3e0e9b39ccd02b
Author: Nicholas Sparks <>
Date:   Mon Dec 17 15:43:11 2018 -0500

    Updated Dot

commit 3779751d4b3c9624ff5bc020722561f2fa13f453
Author: Nicholas Sparks <>
Date:   Mon Dec 17 15:42:55 2018 -0500

    Updated Dot

commit e7421491d46a9b760fc687d677a5b8b4c569fe64
Author: Nicholas Sparks <>
Date:   Mon Dec 17 01:31:49 2018 -0500

    Cleanup

commit 79ab5423738ea18cbfa578781303a8b1caa5d80b
Author: Nicholas Sparks <>
Date:   Mon Dec 17 01:25:52 2018 -0500

    ignore test compile files

commit e047f21a61776da0d29095ecf79056b55ad5ca95
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Mon Dec 17 01:21:52 2018 -0500

    fixed dimention mismatch issue

commit 68ba1bc674b8fbc25359ae572574f5ff1f91dbf2
Merge: 7dfd897 e25bd6c
Author: Nicholas Sparks <>
Date:   Mon Dec 17 00:48:41 2018 -0500

    Dim Fix

commit 7dfd8971096702106fa04fc5dcd357b6e8f04023
Author: Nicholas Sparks <>
Date:   Mon Dec 17 00:44:53 2018 -0500

    Fixed Dim Check
```

```
commit e25bd6ca6c697aacba1e69279764a22c6f5f65d1
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sun Dec 16 23:22:22 2018 -0500

    cleanup and added matrix fail tests

commit b753cc6e838978f2eec8ade8fa0d0994e8fdc335
Author: Nicholas Sparks <>
Date:   Sun Dec 16 13:09:18 2018 -0500

    Demo

commit ad20a47f54d167abfd2ce01a2bf50b87a2de3ced
Merge: cbf57a8 9fa2dcd
Author: Nicholas Sparks <>
Date:   Sun Dec 16 12:51:26 2018 -0500

    Merge branch 'master' of https://github.com/Pearcekieser/CS4115

commit cbf57a81f38f01e6778d47accc4215da66e7db8b
Author: Nicholas Sparks <>
Date:   Sun Dec 16 12:51:14 2018 -0500

    Invalid Dimensions catch

commit 9fa2dcd3361abd7f4a6b44689ca3321ad094a0f9
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sun Dec 16 11:41:19 2018 -0500

    Updated string tests

commit 9c6bc9f791f10ae2f935025de26fe9f6b7a7dafa
Author: Nicholas Sparks <>
Date:   Sun Dec 16 11:13:42 2018 -0500

    Proper naming

commit 2acffc69751f7c456db594e634f539c5506445d7
Author: Nicholas Sparks <>
Date:   Sun Dec 16 11:10:35 2018 -0500

    Change file extension to mx

commit e00f81080a71b20393b13439e756a0d0c3f36d16
Author: Nicholas Sparks <>
Date:   Sun Dec 16 02:55:11 2018 -0500

    fix

commit ea451a3599e5c9b355167515074d27d64547ce73
Author: Nicholas Sparks <>
Date:   Sun Dec 16 02:52:51 2018 -0500

    corrupt test

commit 68d526acaba0ae3c1899dbcd48f08ea9e0b56e84
Author: Nicholas Sparks <>
Date:   Sun Dec 16 02:45:28 2018 -0500

    Declare and set Vars
```

```
commit b7743bcb5f599ce540e6c49e06de31032daad722
Author: Nicholas Sparks <>
Date:   Sat Dec 15 22:44:29 2018 -0500

    Error free make

commit 31e36bf275c74e8b0e0176e6e483a7860f7a0ce5
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sat Dec 15 19:50:48 2018 -0500

    cleaning codebase

commit 1ad9672ab5fc1f659fa9dfcc9ddef4dedf2051e8
Author: Nicholas Sparks <>
Date:   Sat Dec 15 16:44:37 2018 -0500

    more tests

commit db04049edf774cdcb1da5ad6a04f56a336a54dd9
Author: Nicholas Sparks <>
Date:   Sat Dec 15 16:38:22 2018 -0500

    tests

commit 5c66a7a915f8e52c180af04f182b882a435ae91e
Author: Nicholas Sparks <>
Date:   Sat Dec 15 16:37:39 2018 -0500

    added tests

commit 35e719ddf4b7d54a38758447ffbee3022cd3eb2a
Author: Nicholas Sparks <>
Date:   Sat Dec 15 16:24:53 2018 -0500

    Added matrix.c functions

commit 9a0c5d72f8403767c643a7d1cff5d4dc2f4eebc8
Author: Nicholas Sparks <>
Date:   Sat Dec 15 14:45:53 2018 -0500

    printm working

commit 99d35cf20c9909978f9588aac16da02474c76c9b
Author: Nicholas Sparks <>
Date:   Sat Dec 15 14:05:30 2018 -0500

    Codegen update

commit 750ae5efecca8160f4324b78eb4f50e9307c5407
Author: Nicholas Sparks <>
Date:   Sat Dec 15 12:45:57 2018 -0500

    Checks if all types are equal in Matrix

commit 932bce9933dc60a217474877b4f9738a28880449
Author: Nicholas Sparks <>
Date:   Fri Dec 14 14:32:32 2018 -0500

    Some more updates

```

```
commit 154ebd139429b17ce3e6e876d185ba008b2d49de
Author: Nicholas Sparks <>
Date:   Fri Dec 14 12:04:45 2018 -0500

    fixed multi-dimensional Matrix Parsing, & corrupt testall.sh

commit 0faa21e200d4ab9d20723927c05c58cb41871d76
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Thu Dec 13 23:31:24 2018 -0500

    String implementation/functions working and cleaned up string tests

commit 48ae589cb112fdc67ffbfa97377dd98b2641d485
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Thu Dec 13 20:31:28 2018 -0500

    removed old OurParser files

commit 367dce3a7fc065bff994b025c203de53936d8598
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Thu Dec 13 20:23:59 2018 -0500

    fixed matrix.c linking problem

commit c3621943cfb54392a99b7985fa836924890cc4db
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Thu Dec 13 19:58:10 2018 -0500

    some directory cleanup

commit 7dd3b44a84cfc0dbac4c17f48a1cbd68fce754a4
Merge: ea6c5da c6fa85c
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Thu Dec 13 16:46:38 2018 -0500

    Fixed merge conflicts

commit ea6c5da6dd2854b36a0b0a98ad2da4d42e9b8687
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Thu Dec 13 16:38:13 2018 -0500

    WIP string implementation

commit c811c1495e13ce7253b465272221dccde2785e01
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Thu Dec 13 16:36:33 2018 -0500

    Updated semant to accept more than one argument for function decls

commit c6fa85c63b92d5f5d40aea16ee2b0342e2a97304
Merge: f707525 4b284a8
Author: Nicholas Sparks <>
Date:   Thu Dec 13 13:54:32 2018 -0500

    update

commit f7075257fa119f7954a0f9bf67b9ae66d1a4dd43
```

```
Author: Nicholas Sparks <>
Date:   Thu Dec 13 13:52:12 2018 -0500

    Converts MatrixLit 2D to 1D Expr List

commit 4b284a878d7aaf67b07af14fd33b4b955ebb1153
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Thu Dec 13 11:48:01 2018 -0500

    ast print of MatrixLit

commit 2f08766fb73c7e9f912a7e8afea8782bc0613787
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Thu Dec 13 11:41:15 2018 -0500

    Fixed complie

commit 42c132a209c4c1dc3be29162c450df2e5e18e779
Author: Nicholas Sparks <>
Date:   Thu Dec 13 16:36:05 2018 +0000

    update

commit 8f7124d480ed29ecc409584b1a07f3a62b332523
Author: Nicholas Sparks <>
Date:   Thu Dec 13 11:34:47 2018 -0500

    update

commit aa8e4dd9ba539cf314346768267ccf10b1b02968
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Wed Dec 12 20:53:02 2018 -0500

    String declaration and printing now works, tests included

commit 98ace61839555c4f334378d228f65ab04c2e31ba
Author: Nicholas Sparks <>
Date:   Wed Dec 12 17:22:30 2018 -0500

    Fixed printing order of functions

commit 02424f271e8eb86058f497f22160283e5158ad92
Author: Nicholas Sparks <>
Date:   Wed Dec 12 16:43:52 2018 -0500

    updated parser

commit 1c6febf31e8b71d9528d30fe87ebcf79ef52a830
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Dec 11 13:13:40 2018 -0500

    matrix passes scanning

commit e8edc019a559921652c6bcc78fdee00b1682f598
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sun Dec 9 16:08:41 2018 -0500

    fixed: compliler now passes microC tests
```

```

commit 989288cc876c24a2eef5d4d6ff3f06ba475ab3ad
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Dec 4 20:41:31 2018 -0500

    now builds

commit 4c893ad7fe017ace53d22a9d9ea88842335360d4
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sun Dec 2 23:31:24 2018 -0500

    Almost builds on make

commit 248f5f4fb0b2c1fde1f1d40eb561e93ddb8aee2e
Author: Nicholas Sparks <>
Date:   Sat Dec 1 20:02:56 2018 -0500

    Issues with parser, but progress on codegen

commit e7ee115f65356ec5b66e488dabbe3bcd579e6eec
Author: Nicholas Sparks <>
Date:   Fri Nov 30 22:11:41 2018 -0500

    Updated sast, semant. codegen. unbound 'm' Line 146:codegen.ml

commit 12c7590ba13f2b80c1ec544c05d576e9ce39a8a0
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Thu Nov 29 22:33:02 2018 -0500

    Added SMatrixLit processing to codegen expr-builder

commit 2d94ca0f587a9b8bf7536feffd181cad32516775
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Nov 27 11:01:26 2018 -0500

    added printm for matrix to codegen

commit ddfea1e7eaa53891f0365dad251b9f124bf8cbd7
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Mon Nov 26 17:55:51 2018 -0500

    Started linking C lib to codegen

commit 2380b2ffafb03a2eccc5593062950ee3e57462e8
Author: Nicholas Sparks <>
Date:   Thu Nov 22 01:10:15 2018 -0500

    AST/SAST/SEMANT Updated

commit c58cc6b578b2437d50349830a1bfc5f90d728307
Merge: ddad05a 405896e
Author: Nicholas Sparks <>
Date:   Sun Nov 18 05:48:05 2018 -0500

    Merge branch 'master' of https://github.com/Pearcekieser/CS4115

commit ddad05a41d7e3266a445a7baf60bd41c718e43b7
```

```
Author: Nicholas Sparks <>
Date:   Sun Nov 18 05:47:03 2018 -0500

    Not working, but getting closer.

commit 405896ea46a2e8e539c5ba72e833944667ccd09a
Merge: 3d32fb3 08044f6
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 16:57:17 2018 -0500

    Fixed merge conflicts in matrix.c

commit 3d32fb312581ab263bb57a416cc61262c8b112f1
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 16:55:18 2018 -0500

    Finished writing inverse function for matrices

commit 08044f6c5c57c51d05a1120a28f31664f68b527b
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 16:39:44 2018 -0500

    Completed matrix mult and scalar mult functions for matrix

commit 995a9f31e837b801af72c1ff68a0043029b15f6d
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 16:27:20 2018 -0500

    Completed add() function for matrix

commit 9703570bd0c19f1f68e0e9904eabd75522e6282a
Merge: b478e2b e2c3297
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 16:08:05 2018 -0500

    Fixed merge conflicts for matrix.c

commit b478e2bf1ddd8bf2d6c547a03b500e71ef21db77
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 16:01:33 2018 -0500

    Finished determinant function and tests

commit e2c3297ddd1e83c75389d22f691a725940579516
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 16:01:20 2018 -0500

    Finished writing transpose function for matrices

commit 2ca95562988371f81c6694fb0c2e401dc36bf60d
Merge: 8f58e1b defb6ff
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 15:52:40 2018 -0500

    Fixed merge conflicts in matrix.c file

commit 8f58e1bffd811fcd6134d52228256b8385f206b1
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 15:50:35 2018 -0500

    Wrote tranpose function for matrices

```

```
commit defb6ffbafc517f1b0f21056c96727222662b827
Merge: bd2ae8e 7e4dfd5
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 15:49:47 2018 -0500

    Added print function for matrix

commit bd2ae8e4562dcfad02bf037cbf0c8ec382f60cf1
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 15:41:24 2018 -0500

    Continued work on determinant

commit 7e4dfd53d08351f5a980e82aabe6dfe16860cd92
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 15:36:53 2018 -0500

    Finished dot product function and added check for null matrices in constructor

commit c09e7801fa633e49cf175642b7e98dc6495ee793
Merge: aa30f87 5b213c2
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 15:01:00 2018 -0500

    Merge branch 'master' of https://github.com/Pearcekieser/CS4115

commit aa30f87db4a193c54a40e831a4c0f72748772fa4
Author: Julia Sheth <jnsheth@me.com>
Date:   Sat Nov 17 14:59:43 2018 -0500

    Writing dot product function for matrices

commit 5b213c2048b35e2f613bd998fded80362bfd8283
Merge: 1904472 58e1754
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 14:58:44 2018 -0500

    Fixed merge conflicts

commit 19044722fcb65152a382d9168030886da2fcfd02
Author: Katie Pfleger <kjp2157@columbia.edu>
Date:   Sat Nov 17 14:57:33 2018 -0500

    Fixed initMatrx and started determinant

commit 58e17547ed9aeba6fe3ae003dac2842ed7d342eb
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sat Nov 17 14:46:33 2018 -0500

    added matrix.c/die()

commit 304cfe04f65981f849a42f766e63d0fb5eb588cb
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sat Nov 17 14:32:03 2018 -0500

    Added more functions to matrix.c

commit e4faeeb4bc95fd59000b923b0ffed20979843bf7
```

```
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Wed Nov 14 20:11:00 2018 -0500

    added matrix.c

commit f44b2e1ffed885caf113fc0b3b9871230d0db39d
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Wed Nov 14 06:36:18 2018 -0500

    added helloWorldSolution

commit 4077f83600191c36c28dd142a1dcacbbda252a7b
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Nov 13 18:37:59 2018 -0500

    missed a file

commit 7f5d86ac73aee037dc240a3a303b69faf1f30456
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Nov 13 18:36:26 2018 -0500

    more ast matrix fixes

commit 530f0fca305b72cc5ab9519c32959f331fb7c569
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Nov 13 18:31:29 2018 -0500

    fix Matrix type constructor

commit da114fc17c4762cfedd63e1dbfb69590470a7b95
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Tue Nov 13 18:23:51 2018 -0500

    Working on building code

commit c74b35cfedb0b0b239723a6d1925c7b4e7c271f5
Author: alana anderson <afa2132@columbia.edu>
Date:   Tue Nov 13 18:11:09 2018 -0500

    added string to function decls and expressions

commit 46297bf266d692e770b3604b83abd6c1e34730ef
Author: alana anderson <afa2132@columbia.edu>
Date:   Tue Nov 13 18:00:18 2018 -0500

    added string types

commit a39d41aebf5c80f004b8346426d21201541b6022
Author: Nicholas Sparks <43015596+ns3284@users.noreply.github.com>
Date:   Tue Nov 13 04:28:17 2018 +0000

    Updated ast,sast,semant

commit 52d658edd9d27175e7536aceb6e1d1d900c2cec3
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.com>
```

```
Date:   Mon Nov 12 20:47:51 2018 -0500

    added Matrix to AST

commit 6e3321cb660be303c58d6c86ebc398e50f950740
Author: alana anderson <afa2132@columbia.edu>
Date:   Mon Nov 12 20:47:23 2018 -0500

    updating gitignor

commit 143dd92460eb6cbcb39d46e6fe3a068a5c7ee6db
Author: ns3284@columbia.edu <43015596+ns3284@users.noreply.github.
    ↪ com>
Date:   Tue Nov 13 00:09:25 2018 +0000

    Updated Parser

commit aab164a9320b49ae525386d232b9d9bd6a9d2f15
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
    ↪ com>
Date:   Sun Nov 11 17:04:49 2018 -0500

    codgen with matrix

commit a510db3e7948f7106ed4dbaf1906a389a5d1380c
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Sat Nov 10 02:55:31 2018 -0500

    Updating  Tokenizer, Parser, AST, SAST

commit d50075abc51fabdb312582cf865e4bfe5d02fce9
Author: alana anderson <afa2132@columbia.edu>
Date:   Fri Nov 9 08:17:38 2018 -0500

    took string out of parser

commit ec252eb258c583c4494e5063c9bb18b4a40971f1
Author: Julia Sheth <jnsheth@me.com>
Date:   Thu Nov 8 16:58:03 2018 -0500

    Rewriting parser so that it conforms more to MicroC and
        ↪ starting to include matrix functionality

commit 003aa4c9f9c6b1819f6bfebe42513872220125be
Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
    ↪ com>
Date:   Thu Nov 8 10:48:07 2018 -0500

    reorganize code and added todo list

commit 086b5a305d53cdaa6091ec4b746a099e8666b746
Author: alana anderson <afa2132@columbia.edu>
Date:   Wed Nov 7 18:12:50 2018 -0500

    wrote basic top-level

commit 047fb1089892322c5450c035c49f3913c5f8080a
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Mon Oct 15 14:21:05 2018 -0400

    forgot ID.method() and nested bool_seqs

```

```
commit 5a9ed02444ebacb8795b774b35b267fefab1fe3d
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Sat Oct 13 23:52:37 2018 -0400

    int -> ref

commit a7e4d7a3d4c2419d8f130e1175456a040c3cce4c
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Sat Oct 13 23:47:48 2018 -0400

    Can't forgot conditionals

commit 8753f9620167291e807ab54709217130259405ba
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Sat Oct 13 23:15:15 2018 -0400

    Corrected the type of MATRIX

commit b2fdccb25c2f35912c332019a4e00bcdf5ecd991
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Sat Oct 13 23:11:01 2018 -0400

    Parser compiles without warning. Untested.

commit 293362985ffa2bec765583dbb4af06d061e748f5
Author: Pearce kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Fri Oct 12 13:08:16 2018 -0400

    Added MicroC

commit 09c39facf88a8584bfe8d2d1fbffc62a8bdc4300
Author: Nicholas Sparks <ns3284@columbia.edu>
Date:   Mon Oct 8 00:32:42 2018 -0400

    Updated tokenizer, parser.mly

commit dc28500c9edf0eb904b648cb232113eaa9d49552
Author: Pearce kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sat Oct 6 19:09:33 2018 -0400

    tokenizer added

commit 979f256397eee25bd4d245bc64453f54a79bb739
Author: ns3284@columbia.edu <ns3284@columbia.edu>
Date:   Sun Oct 7 06:04:42 2018 -0400

    Added conditions, and scope

commit b1d613d76048b83ce6f579e810547ba818c41d30
Author: Pearce kieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sat Oct 6 13:30:45 2018 -0400

    added calc.ml

commit 3a5d9a9ae33a81ce3f742bc14d5298e37982b471
Author: Pearcekieser <5055971+Pearcekieser@users.noreply.github.com>
Date:   Sat Oct 6 13:14:40 2018 -0400

```

```
    Add files via upload

commit 6fb63efd49502f0cc5cb4bd455ae91e9d2ad7219
Author: Pearcekieser <5055971+Pearcekieser@users.noreply.github.com
    ↪ >
Date:   Sat Sep 15 17:39:10 2018 -0400

    Update README.md

commit 95b95fe80e609ae999428fe280855b924a4eb7d7
Author: Pearcekieser <5055971+Pearcekieser@users.noreply.github.com
    ↪ >
Date:   Sat Sep 15 17:39:00 2018 -0400

    Initial commit
```

8.1.2 scanner.mll

```
(* Ocamllex scanner for Matrx *)
{
    module Lex = Lexing
    module Buf = Buffer

let sprintf  = Printf.sprintf

let position lexbuf =
    let p = lexbuf.Lex.lex_curr_p in
        sprintf "%s:%d:%d"
        p.Lex.pos_fname p.Lex.pos_lnum (p.Lex.pos_cnum - p.
            ↪ Lex.pos_bol)

let set_filename (fname:string) (lexbuf:Lex.lexbuf)  =
    ( lexbuf.Lex.lex_curr_p <-
        { lexbuf.Lex.lex_curr_p with Lex.pos_fname = fname }
    ; lexbuf
    )

open Parser
}

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

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"      { comment lexbuf }           (* Comments *)
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['       { LBRACK }
| ']'       { RBRACK }
| ';'       { SEMI }
| ','       { COMMA }
| '+'       { PLUS }
```

```
| '-'        { MINUS }
| '*'        { TIMES }
| '/'        { DIVIDE }
| '='        { ASSIGN }
| "=="       { EQ }
| "!="       { NEQ }
| '<'        { LT }
| "<="       { LEQ }
| ">"        { GT }
| ">="       { GEQ }
| "&&"       { AND }
| "||"       { OR }
| "!"        { NOT }
| "if"       { IF }
| "else"     { ELSE }
| "for"      { FOR }
| "while"    { WHILE }
| "return"   { RETURN }
| "int"      { INT }
| "bool"     { BOOL }
| "float"    { FLOAT }
| "string"   { STRING }
| "void"     { VOID }
| "matrix"   { MATRIX }
| '"'        { STRINGLIT ( string (Buf.create 100) lexbuf ) }
| '\''       { CHARLIT ( char (Buf.create 100) lexbuf) }
| "true"     { BLIT(true)  }
| "false"    { BLIT(false) }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLOATLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and string buf = parse
| [^'"' '\n' '\\']+ as content { Buf.add_string buf content ; string buf lexbuf }
| '\n'      { Buf.add_string buf "\n"; Lex.new_line lexbuf; string buf lexbuf }
| '\\' '"'  { Buf.add_char buf '"'; string buf lexbuf }
| '\\'      { Buf.add_char buf '\\'; string buf lexbuf }
| '"'       { Buf.contents buf } (* return *)

and char buf = parse
| [^'\\'] as content '\'' { content }
| '\\' '\\' '\'' { '\\' }
| '\\' '?' '\'' { '?' }
| '\\' '\'' '\'' { '\'' }
| '\\' '\"' '\'' { '\"' }
| '\\' 'a' '\'' { Char.chr 7 }
| '\\' 'b' '\'' { Char.chr 8 }
```

```
| '\\' 'e' '\'' { Char.chr 27 }
| '\\' 'f' '\'' { Char.chr 12 }
| '\\' 'n' '\'' { Char.chr 10 }
| '\\' 'r' '\'' { Char.chr 13 }
| '\\' 't' '\'' { Char.chr 9 }
| '\\' 'v' '\'' { Char.chr 11 }


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

8.1.3 parser.mly

```
%{
  open Ast
%}

%token MATRIX STRING CHAR LBRACK RBRACK
%token FLOAT /* STRING */
%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLOATLIT
%token <string> STRINGLIT
%token <char> CHARLIT
%token EOF

%start program
%type <Ast.program> program

%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

%%


program:
  decls EOF { 1 decls:/* nothing */  [], [] | decls vdecl  ((2 :: fst1),
      snd 1)|declsfdecl(fst1, (2 :: snd1)) fdecl:typ ID LPAREN
      formals_optRPARENLBRACEvdecl_liststmt_listRBRACEtyp =1;fname =
      2;formals = List.rev4;locals = List.rev 7;body = List.rev8
```

formals$_{o}pt : /*nothing*/[]|formal_{l}ist1$ formal$_{l}ist :$ $typID[(1,2, Noexpr)]|formal_{l}istCOMMAtypID(3,4, Noexpr) ::1$ typ:INT Int | BOOL Bool | FLOAT Float | STRING String | CHAR Char | MATRIX Matrix | VOID Void vdecl$_{l}ist : /*nothing*/[]|vdecl_{l}istvdecl2$ :: $1vdecl : typIDSEMI(1, 2, Noexpr)|typIDASSIGNexprSEMI(1, 2, Asn(2,4))stmt_{l}ist : /*nothing*/[]|stmt_{l}iststmt2$ :: $1stmt : exprSEMIExpr(1)$ | RETURN expr$_{o}ptSEMIReturn(2)$ | LBRACE stmt$_{l}istRBRACEBlock(List.rev2)$ | IF LPAREN expr RPAREN stmt | IF LPAREN expr RPAREN stmt ELSE stmt If(3,5, $7)|FORLPARENexpr_{o}ptSEMIexprSEMIexpr_{o}ptRPARENstmtFor(3, 5,7, 9)|WHILELPARENexprRPARENstmtWhile(3, 5)expr_{o}pt : /*nothing*/Noexpr|expr1$ expr:LITERAL Literal(1)$|FLOATLITFliteral(1)$ | BLIT BoolLit(1)$|CHARLITCharLit(1)$ | STRINGLIT StringLit(1)$|TRUEBoolLit(true)|FALSEBoolLit(false)|IDId(1)$ /* | ID matrx$_{a}ddrGetMatrixVal(1,2)*//*$ $|IDmatrx_{a}ddrASSIGNexprSetMatrixVal(1,2,4)$ */| LBRACK matrx$_{v}alueRBRACKMatrixLit(2)$ | 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 | NOT expr Unop(Not, $2)|IDASSIGNexprAsn(1, 3)/*handlesassignment*/|IDLPARENseq_{o}ptRPARENCall(1, 3)|LPARENexprRPAREN2$ seq$_{o}pt : /*nothing*/[]|seqList.rev1$ seq:expr $[1]|seqCOMMAexpr3$ :: $1matrx_{v}alue : LBRACKseqRBRACK[MatrixLit(List.rev2)]$ | LBRACK seq RBRACK COMMA matrx$_{v}alueMatrixLit(List.rev2)::5|LBRACKseqRBRACKmatrx_{v}alueMatrixLit(List.rev2)::4$

8.1.4 ast.ml

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

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

type uop = Neg | Not

type expr =
    Literal of int
  | Fliteral of string
  | BoolLit of bool
  | CharLit of char
  | StringLit of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Asn of string * expr
  | Call of string * expr list
  | Noexpr
  | MatrixLit of expr list

type typ = Int | Bool | Float | Void | String | Char |
    ↪ Matrix

```

```
type bind = typ * string * expr

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

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

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | CharLit(l) -> Char.escaped l
  | StringLit(l) -> l
  | MatrixLit(l) -> "matrixLit[" ^ String.concat ", " (List.
      map string_of_expr l) ^ "]"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
          string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
```

```
  | Asn(v, e) -> v ^ " = " ^ string_of_expr e
  | 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
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | String -> "string"
  | Char -> "char"
  | Matrix -> "matrix"
  | Void -> "void"

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

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map (fun (_, vName, _) -> vName) 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) =
  let f' = List.rev funcs in
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl f')
```

8.1.5 sast.ml

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

```

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
  | SCharLit of char
  | SStringLit of string
  | SId of string
  (* list of contents, number of rows then cols *)
  | SMatrixLit of sexpr list * int * int
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAsn of string * sexpr
  | SCall of string * sexpr list
  | SNoexpr

type sbind = typ * string * sexpr

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

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

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFliteral(l) -> l
  | SCharLit(l) -> Char.escaped l
  | SStringLit(l) -> l
  | SMatrixLit(l, r, c) -> "rows: " ^ string_of_int r ^ ", cols: " ^ string_of_int c ^ " : [" ^ String.concat ", " (List.map string_of_sexpr l) ^ "]"
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
```

```
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAsn(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SNoexpr -> ""
      ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map (fun (_, vName, _) -> vName) fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (vars, funcs) =
  let f' = List.rev funcs in
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl f')
```

8.1.6 semant.ml

```

(* Semantic checking for the Matrx compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

```

```
(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

  (* Verify a list of bindings has no void types or duplicate names *)
  let check_binds (kind : string) (binds : bind list) =
    List.iter (function
 (Void, b, _) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      | ((_,n1,_) :: (_,n2,_) :: _) when n1 = n2 ->
   raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a,_) (_,b,_) -> compare a b) binds)
  in

  (**** Check global variables ****)

  check_binds "global" globals;

  (**** Check functions ****)

  (* Collect function declarations for built-in functions: no bodies *)
  let built_in_decls =
    let add_bind map (name, ty, ret) = StringMap.add name {
    typ = ret;
    fname = name;
    formals =
      (let rec create_ty_list = (function
        [] -> []
        | hd::tl -> (hd, "x", Noexpr)::(create_ty_list tl))
      in create_ty_list ty);
    locals = []; body = [] } map
  in List.fold_left add_bind StringMap.empty [ ("print", [Int], Void);
                               ("printstr", [String], Void);
                               ("printm", [Matrix], Void);
                               ("printb", [Bool], Void);
                               ("printf", [Float], Void);
                               ("length", [String], Int);
                               ("transpose", [Matrix], Matrix);
                               ("matmult", [Matrix; Matrix], Matrix);
```

```
                ("matadd", [Matrix; Matrix], Matrix);
                ("dot", [Matrix; Matrix], Matrix);
                ("matscale", [Matrix; Int], Matrix);
                ("det", [Matrix; Int], Int);
                ("get_char", [String; Int], String);
                ("sconcat", [String; String], String);
                ("sequals", [String; String], Bool);
                ("substring", [String; Int; Int], String)]
  in

  (* Add function name to symbol table *)
  let add_func map fd =
   let built_in_err = "function " ^ fd.fname ^ " may not be defined"
   and dup_err = "duplicate function " ^ fd.fname
   and make_err er = raise (Failure er)
   and n = fd.fname (* Name of the function *)
   in match fd with (* No duplicate functions or redefinitions of built-ins *)
       _ when StringMap.mem n built_in_decls -> make_err built_in_err
     | _ when StringMap.mem n map -> make_err dup_err
     | _ ->  StringMap.add n fd map
  in

  (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls functions
  in

  (* Return a function from our symbol table *)
  let find_func s =
   try StringMap.find s function_decls
   with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

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

  let check_function func =
   (* Make sure no formals or locals are void or duplicates *)
   check_binds "formal" func.formals;
   check_binds "local" func.locals;

   (* Raise an exception if the given rvalue type cannot be assigned to
```

```
      the given lvalue type *)
   let check_assign lvaluet rvaluet err =
      if lvaluet = rvaluet then lvaluet else raise (Failure err)
   in

   let rec get_dims = function
       MatrixLit l -> List.length l :: get_dims (List.hd l)
     | _ -> []
   in
   (* Raise an exception if dimensions of Matrix are not balanced *)
   let rec flatten d = function
     [] -> []
     | MatrixLit hd::tl -> if List.length hd != List.hd d then raise (Failure("Invalid dims")) else List.append (flatten (List.tl d) hd) (flatten d tl)
     | a -> a
   in
   (* Build local symbol table of variables for this function *)
   let symbols = List.fold_left (fun m (ty, name, _) -> StringMap.add name ty m)
                StringMap.empty (globals @ func.formals @ func.locals )
   in

   (* Return a variable from our local symbol table *)
   let type_of_identifier s =
     try StringMap.find s symbols
     with Not_found -> raise (Failure ("undeclared identifier " ^ s))
   in


   (* Return a semantically-checked expression, i.e., with a type *)
   let rec expr = function
       Literal  l -> (Int, SLiteral l)
     | Fliteral l -> (Float, SFliteral l)
     | BoolLit l  -> (Bool, SBoolLit l)
     | CharLit l ->  (Char, SCharLit l)
     | StringLit l  -> (String, SStringLit l)
     | MatrixLit l ->
         let d = get_dims (MatrixLit l) in
         let rec all_match = function
           [] -> ignore()
           | hd::tl -> if tl != [] then
                         let (t1, _) = expr hd in let (t2, _) = expr (List.hd tl) in
                         if t1 = t2 then all_match tl else raise (Failure ("Data Mismatch in MatrixLit: " ^ string_of_typ t1 ^ " does
```

```
                                 not match " ^ string_of_typ t2))
                        else ignore()
            in
            all_match l;
            if List.length d > 2 then (Matrix, SMatrixLit ((List.map expr l), List.hd d, List.hd (List.tl d)))
            else if List.length d = 2 then (Matrix, SMatrixLit ( (List.map expr (flatten (List.tl d) l)), List.hd d, List.hd (List.tl d)))
            else if List.length d = 1 then (Matrix, SMatrixLit ( (List.map expr (flatten (List.tl d) l)), List.hd d, 1))
            else (Matrix, SMatrixLit ( (List.map expr l), 0,0) )
          | Noexpr     -> (Void, SNoexpr)
          | Id s       -> (type_of_identifier s, SId s)
          | Asn(var, e) as ex ->
            let lt = type_of_identifier var
            and (rt, e') = expr e in
            let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
              string_of_typ rt ^ " in " ^ string_of_expr ex
            in (check_assign lt rt err, SAsn(var, (rt, e')))
          | Unop(op, e) as ex ->
            let (t, e') = expr e in
            let ty = match op with
              Neg when t = Int || t = Float -> t
            | Not when t = Bool -> Bool
            | _ -> raise (Failure ("illegal unary operator " ^
                                   string_of_uop op ^ string_of_typ t ^
                                   " in " ^ string_of_expr ex))
            in (ty, SUnop(op, (t, e')))
          | Binop(e1, op, e2) as e ->
            let (t1, e1') = expr e1
            and (t2, e2') = expr e2 in
            (* All binary operators require operands of the same type *)
            let same = t1 = t2 in
            (* Determine expression type based on operator and operand types *)
            let ty = match op with
              Add | Sub | Mult | Div when same && t1 = Int -> Int
            | Add | Sub | Mult | Div when same && t1 = Float -> Float
            | Equal | Neq            when same -> Bool
            | Less | Leq | Greater | Geq
                       when same && (t1 = Int || t1 = Float) -> Bool
```

```
          | And | Or when same && t1 = Bool -> Bool
          | _ -> raise (
      Failure ("illegal binary operator " ^
                        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                        string_of_typ t2 ^ " in " ^ string_of_expr e))
          in (ty, SBinop((t1, e1'), op, (t2, e2')))
      | Call(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.formals in
          if List.length args != param_length then
            raise (Failure ("expecting " ^ string_of_int param_length ^
                             " arguments in " ^ string_of_expr call))
          else let check_call (ft, _) e =
            let (et, e') = expr e in
            let err = "illegal argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
            in (check_assign ft et err, e')
          in
          let fdFormals = List.map (fun (tp, vName, _) -> ( tp, vName) ) fd.formals in
          let args' = List.map2 check_call fdFormals args
          in (fd.typ, SCall(fname, args'))
  in

  let check_bool_expr e =
    let (t', e') = expr e
    and err = "expected Boolean expression in " ^ string_of_expr e
    in if t' != Bool then raise (Failure err) else (t', e')
  in

  (* Return a semantically-checked statement i.e. containing sexprs *)
  let rec check_stmt = function
      Expr e -> SExpr (expr e)
    | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1, check_stmt b2)
    | For(e1, e2, e3, st) ->
 SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
    | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
    | Return e -> let (t, e') = expr e in
      if t = func.typ then SReturn (t, e')
      else raise (
 Failure ("return gives " ^ string_of_typ t ^ " expected " ^
   string_of_typ func.typ ^ " in " ^ string_of_expr e))
```

```

      (* A block is correct if each statement is correct and nothing
         follows any Return statement.  Nested blocks are flattened. *)
       | Block sl ->
           let rec check_stmt_list = function
               [Return _ as s] -> [check_stmt s]
             | Return _ :: _   -> raise (Failure "nothing may follow a return")
             | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten blocks *)
             | s :: ss         -> check_stmt s :: check_stmt_list ss
             | []              -> []
           in SBlock(check_stmt_list sl)

     in (* body of check_function *)
     { styp = func.typ;
       sfname = func.fname;
       sformals = List.map (fun (t,n,v) -> (t,n, expr v)) func.formals;
       slocals  = List.map (fun (t,n,v) -> (t,n, expr v)) func.locals;
       sbody = match check_stmt (Block func.body) with
 SBlock(sl) -> sl
       | _ -> raise (Failure ("internal error: block didn't become a block?"))
     }
   in (globals, List.map check_function functions)
```

8.1.7 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 Sast

```

```
module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context    = L.global_context () in
  let llmem = L.MemoryBuffer.of_file "matrix.bc" in
  let llm = Llvm_bitreader.parse_bitcode context llmem in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "Matrx" in

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  and void_t     = L.void_type   context
  and string_t   = L.pointer_type (L.i8_type context)
  and matrx_t    = L.pointer_type (match L.type_by_name llm "struct.matrix" with
      None -> raise (Failure "Missing implementation for struct Matrix")
    | Some t -> t)
  in

  (* Return the LLVM type for a Matrx type *)
  let ltype_of_typ = function
      A.Int    -> i32_t
    | A.Bool   -> i1_t
    | A.Float  -> float_t
    | A.Void   -> void_t
    | A.Char   -> i8_t
    | A.String -> string_t
    | A.Matrix -> matrx_t
  in

  (* Create a map of global variables after creating each *)
  let global_vars : L.llvalue StringMap.t =
    let global_var m (t, n, _) =
      let init = match t with
          A.Float -> L.const_float (ltype_of_typ t) 0.0
        | _ -> 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

  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
    L.declare_function "printf" printf_t the_module in

  let printMatrix_t = L.function_type i32_t [| matrx_t |] in
```

```
    let printMatrix_f = L.declare_function "display" printMatrix_t the_module in
    let matrix_init_t = L.function_type matrx_t [|i32_t ; i32_t|] in
    let matrix_init_f = L.declare_function "initMatrix_CG" matrix_init_t the_module in
    let store_matrix_t = L.function_type matrx_t [|matrx_t ; i32_t |] in
    let store_matrix_f = L.declare_function "storeVal" store_matrix_t the_module in
    let transpose_matrix_t = L.function_type matrx_t [|matrx_t |] in
    let transpose_matrix_f = L.declare_function "transpose" transpose_matrix_t the_module in
    let mult_matrix_t = L.function_type matrx_t [|matrx_t; matrx_t|] in
    let mult_matrix_f = L.declare_function "matrixMult" mult_matrix_t the_module in
    let add_matrix_t = L.function_type matrx_t [|matrx_t; matrx_t|] in
    let add_matrix_f = L.declare_function "mAdd" add_matrix_t the_module in
    let dot_matrix_t = L.function_type matrx_t [|matrx_t; matrx_t|] in
    let dot_matrix_f = L.declare_function "dotProduct" dot_matrix_t the_module in
    let scale_matrix_t = L.function_type matrx_t [|matrx_t ; i32_t |] in
    let scale_matrix_f = L.declare_function "timesScalar" scale_matrix_t the_module in
    let det_matrix_t = L.function_type matrx_t [|matrx_t ; i32_t |] in
    let det_matrix_f = L.declare_function "determinant" det_matrix_t the_module in


    let string_get_t = L.function_type string_t [| string_t; i32_t |] in
    let string_get_f = L.declare_function "string_get" string_get_t the_module in
    let string_length_t = L.function_type i32_t [| string_t |] in
    let string_length_f = L.declare_function "string_length" string_length_t the_module in
    let string_concat_t = L.function_type string_t [| string_t ; string_t |] in
    let string_concat_f = L.declare_function "string_concat" string_concat_t the_module in
    let string_equals_t = L.function_type i32_t [| string_t; string_t |] in
    let string_equals_f = L.declare_function "string_equals" string_equals_t the_module in
    let string_substr_t = L.function_type string_t [| string_t ; i32_t; i32_t |] in
```

```
  let string_substr_f = L.declare_function "string_substr" string_substr_t the_module in

    (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types =
 Array.of_list (List.map (fun (t,_,_) -> ltype_of_typ t) fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.sfname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in
    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
    let float_format_str = L.build_global_stringptr "%g\n" "fmt" builder in
    let string_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
    (* Construct the function's "locals": formal arguments and locally
       declared variables.  Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the "locals" map *)
    let local_vars =
      let add_formal m (t, n) p =
        L.set_value_name n p;
 let local = L.build_alloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
 StringMap.add n local m

      (* Allocate space for any locally declared variables and add the
       * resulting registers to our map *)
      and add_local m (t, n) =
  let local_var = L.build_alloca (ltype_of_typ t) n builder
 in StringMap.add n local_var m
      in

```

```
      let sformals = List.map (fun (tp, vName, _) -> (tp, vName)) fdecl.sformals in
      let slocals= List.map (fun (tp, vName, _) -> (tp, vName)) fdecl.slocals in
      let formals = List.fold_left2 add_formal StringMap.empty sformals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals slocals
    in

    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder ((_, e) : sexpr) = match e with
    SLiteral i  -> L.const_int i32_t i
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
      | SFliteral i -> L.const_float float_t (float_of_string i)
      | SCharLit l  -> L.const_int i8_t (int_of_char l)
      | SStringLit l -> L.build_global_stringptr l "tmp" builder
      | SNoexpr     -> L.const_int i32_t 0
      | SId s       -> L.build_load (lookup s) s builder
      | SMatrixLit (contents, rows, cols) ->
              let rec expr_list = function
                [] -> []
                | hd::tl -> expr builder hd::expr_list tl
              in
              let contents' = expr_list contents
              in
              let m = L.build_call matrix_init_f [| L.const_int i32_t cols; L.const_int i32_t rows |] "matrix_init" builder
              in
              ignore(List.map (fun v -> L.build_call store_matrix_f [| m ; v |] "store_val" builder) contents'); m
      | SAsn (s, e) -> let e' = expr builder e in
                        ignore(L.build_store e' (lookup s) builder); e'
      | SBinop ((A.Float,_ ) as e1, op, e2) ->
   let e1' = expr builder e1
   and e2' = expr builder e2 in
   (match op with
     A.Add     -> L.build_fadd
   | A.Sub     -> L.build_fsub
   | A.Mult    -> L.build_fmul
   | A.Div     -> L.build_fdiv
   | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
```

```
    | A.Neq     -> L.build_fcmp L.Fcmp.One
    | A.Less    -> L.build_fcmp L.Fcmp.Olt
    | A.Leq     -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Geq     -> L.build_fcmp L.Fcmp.Oge
    | A.And | A.Or ->
        raise (Failure "internal error: semant should have rejected and/or on float")
    ) e1' e2' "tmp" builder
       | SBinop (e1, op, e2) ->
    let e1' = expr builder e1
    and e2' = expr builder e2 in
    (match op with
      A.Add     -> L.build_add
    | A.Sub     -> L.build_sub
    | A.Mult    -> L.build_mul
           | A.Div     -> L.build_sdiv
    | A.And     -> L.build_and
    | A.Or      -> L.build_or
    | A.Equal   -> L.build_icmp L.Icmp.Eq
    | A.Neq     -> L.build_icmp L.Icmp.Ne
    | A.Less    -> L.build_icmp L.Icmp.Slt
    | A.Leq     -> L.build_icmp L.Icmp.Sle
    | A.Greater -> L.build_icmp L.Icmp.Sgt
    | A.Geq     -> L.build_icmp L.Icmp.Sge
    ) e1' e2' "tmp" builder
       | SUnop(op, ((t, _) as e)) ->
           let e' = expr builder e in
    (match op with
      A.Neg when t = A.Float -> L.build_fneg
    | A.Neg                  -> L.build_neg
           | A.Not                  -> L.build_not) e' "tmp" builder
       | SCall ("print", [e]) | SCall ("printb", [e]) ->
        L.build_call printf_func [| int_format_str ; (expr builder e) |] "printf" builder
       | SCall ("printstr", [e]) ->
         L.build_call printf_func [| string_format_str ; (expr builder e) |] "printf" builder
       | SCall ("printf", [e]) ->
        L.build_call printf_func [| float_format_str ; (expr builder e) |] "printf" builder
       | SCall ("printm", [e]) ->
         L.build_call printMatrix_f [| (expr builder e) |] "printm" builder
       | SCall ("transpose", [e]) ->
         L.build_call transpose_matrix_f [| (expr builder e) |] "transpose" builder
       | SCall ("matmult", [e1; e2]) ->
         L.build_call mult_matrix_f [| (expr builder e1); (expr builder e2) |] "matmult" builder
       | SCall ("matadd", [e1; e2]) ->
         L.build_call add_matrix_f [| (expr builder e1); (expr builder e2) |] "matadd" builder
```

```
      | SCall ("dot", [e1; e2]) ->
        L.build_call dot_matrix_f [| (expr builder e1); (expr builder e2) |] "dot" builder
      | SCall ("matscale", [e1; e2]) ->
        L.build_call scale_matrix_f [| (expr builder e1); (expr builder e2) |] "matscale" builder
      | SCall ("det", [e1; e2]) ->
        L.build_call det_matrix_f [| (expr builder e1); (expr builder e2) |] "det" builder
      | SCall ("length", [e]) ->
        L.build_call string_length_f [| (expr builder e) |] "string_length" builder
      | SCall ("get_char", [e;index]) ->
                let index = expr builder index in
                let e = expr builder e in
                L.build_call string_get_f [| e; index |] "string_get" builder;
      | SCall("sconcat", [e1; e2]) ->
         L.build_call string_concat_f [| (expr builder e1); (expr builder e2) |] "string_concat" builder
      | SCall("sequals", [e1; e2]) ->
         L.build_call string_equals_f [| (expr builder e1); (expr builder e2) |] "string_equals" builder
      | SCall("substring", [s; e1; e2]) ->
         L.build_call string_substr_f [| (expr builder s); (expr builder e1); (expr builder e2) |]
         "string_substr" builder
      | SCall (f, args) ->
         let (fdef, fdecl) = StringMap.find f function_decls in
let llargs = (List.rev (List.map (expr builder) (List.rev args))) in
let result = (match fdecl.styp with
                        A.Void -> ""
                      | _ -> f ^ "_result") in
         L.build_call fdef (Array.of_list llargs) result builder
  in
  ignore(List.map (fun (_, _, v) -> expr builder v) fdecl.sformals);
  ignore(List.map (fun (_, _, v) -> expr builder v) fdecl.slocals);

  (* LLVM insists each basic block end with exactly one "terminator"
     instruction that transfers control.  This function runs "instr builder"
     if the current block does not already have a terminator.  Used,
     e.g., to handle the "fall off the end of the function" case. *)
  let add_terminal builder instr =
    match L.block_terminator (L.insertion_block builder) with
```

```
Some _ -> ()
     | None -> ignore (instr builder) in

   (* Build the code for the given statement; return the builder for
      the statement's successor (i.e., the next instruction will be built
      after the one generated by this call) *)

   let rec stmt builder m = function
 SBlock sl ->
              let helper (bldr, map) = stmt bldr map in
              let (b, _) = List.fold_left helper (builder, m ) sl in
              (b, m)
     | SExpr e -> ignore(expr builder e); (builder, m)
     | SReturn e -> ignore(match fdecl.styp with
                                (* Special "return nothing" instr *)
                                A.Void -> L.build_ret_void builder
                                (* Build return statement *)
                              | _ -> L.build_ret (expr builder e) builder );
                        (builder, m)
     | SIf (predicate, then_stmt, else_stmt) ->
       let bool_val = expr builder predicate in
 let merge_bb = L.append_block context "merge" the_function in
       let build_br_merge = L.build_br merge_bb in (* partial function *)

  let then_bb = L.append_block context "then" the_function in
  let (b', _) = stmt (L.builder_at_end context then_bb) m then_stmt in
  add_terminal b'
  build_br_merge;

 let else_bb = L.append_block context "else" the_function in
  let (b', _) = stmt (L.builder_at_end context else_bb) m else_stmt in
  add_terminal b'
   build_br_merge;

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

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

```

```
    let body_bb = L.append_block context "while_body"
        ↪ the_function in
     let (b', _) = stmt (L.builder_at_end context body_bb) m
         ↪ body in
     add_terminal b'
      (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, m)

       (* Implement for loops as while loops *)
       | SFor (e1, e2, e3, body) -> stmt builder m
      ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr
          ↪ e3]) ] )
     in

     (* Build the code for each statement in the function *)
     let (builder, _) = stmt builder local_vars (SBlock fdecl
         ↪ .sbody) in

     (* Add a return if the last block falls off the end *)
     add_terminal builder (match fdecl.styp with
         A.Void -> L.build_ret_void
       | A.Float -> L.build_ret (L.const_float float_t 0.0)
       | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

8.1.8 matrx.ml

```
(* Top-level of the Matrx compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it,
       ↪ generate LLVM IR,
   and dump the module *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the
       ↪ generated LLVM IR");
```

```
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./matrx.native [-a|-s|-l|-c] [file.mc]" in (* changed name to matrx *)
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in (* changed name to Parser *)
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
 Llvm_analysis.assert_valid_module m;
 print_string (Llvm.string_of_llmodule m)
```

8.1.9 matrix.c

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

static void die(const char *message)
{
    perror(message);
    exit(1);
}

struct matrix {
  int num_rows;
  int num_cols;
  int** matrixAddr; // accessed [row][col]
  int buildPosition;
};
typedef struct matrix matrix;

int debug = 0;



matrix* storeVal(matrix* target, int value) {
    int position = target->buildPosition;
    int curr_row = position / target->num_cols;
    int curr_col = position % target->num_cols;

    if(debug == 1) {
        printf("Storing: %d\n", value);
        printf("in row: %d\n", curr_row);
        printf("in col: %d\n", curr_col);
```

```
    }

    target->matrixAddr [curr_row][curr_col] = value;
    target->buildPosition = target->buildPosition + 1;
    return target;
}

matrix* initMatrix(int* listOfValues, int num_cols, int num_rows) {
  int** matrixValues = malloc(num_rows * sizeof(int*));

  if(debug == 1) {
      printf("Building matrix:\n");
      printf("num_rows: %d\n", num_rows);
      printf("num_cols: %d\n", num_cols);
  }

  //set all values in matrix to NULL if list of values is NULL
  if (listOfValues == NULL) {
    for(int i = 0; i < num_rows; i++) {
      int* matrix_row = malloc(num_cols * sizeof(int));
      *(matrixValues + i) = matrix_row;
      for(int j = 0; j < num_cols; j++) {
        matrix_row[j] = 0;
      }
    }
  }

  //load values from a list of values
  else {
    for(int i = 0; i < num_cols; i++) {
      int* matrix_col = malloc(num_rows * sizeof(int));
      *(matrixValues + i) = matrix_col;
      for(int j = 0; j < num_rows; j++) {
        matrix_col[j] = listOfValues[i*num_rows + j];
      }
    }
  }

  //return a pointer to matrix struct
  matrix* result = malloc(sizeof(struct matrix));
  result->num_cols = num_cols;
  result->num_rows = num_rows;
  result->matrixAddr = matrixValues;
  result->buildPosition = 0;
  return result;
}

matrix* initMatrix_CG( int num_cols, int num_rows) {
    return initMatrix(NULL, num_cols, num_rows);
}

matrix* mAdd(matrix* lhs, matrix* rhs) {
  //check dimensions
  if (lhs->num_rows != rhs->num_rows || lhs->num_cols != rhs->num_rows) {
    die("matrix add size mismatch");
  }
  int rows = lhs->num_rows;
  int cols= lhs->num_cols;
  matrix *result = initMatrix(NULL, rows, cols);
  for(int i=0; i<rows; i++) {
    for(int j=0; j<cols; j++) {
```

```
            int sum = lhs->matrixAddr[i][j] + rhs->matrixAddr[i][j];
            result->matrixAddr[i][j] = sum;
        }
    }

    return result;
}


void getCofactor(matrix* m, matrix* temp, int p, int q, int n) {
    int i=0, j=0;
    for(int row=0; row<n; row++) {
        for(int col=0; col<n; col++) {
            if(row != p && col != q) {
                temp->matrixAddr[i][j++] = m->matrixAddr[row][col];
                if(j == n-1) {
                    j=0;
                    i++;
                }
            }
        }
    }
}


int determinant(matrix* input, int n) {
    int row = input->num_rows;
    int col = input->num_cols;
    int d = 0;

    if(row==col) {
        //base case: matrix contains single element
        if(n == 1)
            return input->matrixAddr[0][0];

        matrix* temp = initMatrix(NULL, row, col);

        int sign=1;
        for(int f=0; f<n; f++) {
            getCofactor(input, temp, 0, f, n);
            d += sign * input->matrixAddr[0][f] * determinant(temp,
                n-1);
            sign = -sign;
        }

        return d;

    }
    else {
        printf("Your matrix must be square to compute the
            determinant.\n");
        return 0;
    }

}


matrix* dotProduct(matrix* lhs, matrix* rhs) {
    //check to make sure matrices are the same size
    if (lhs->num_cols != rhs->num_rows) {
        die("Matrices are not the same size!");
    }
```

```
    //once we know that matrices are same size, we can compute result
    matrix *result = initMatrix(NULL, rhs->num_cols, lhs->num_rows);
    for (int i=0; i < lhs->num_rows; i++)
    {
        for (int j=0; j < rhs->num_cols; j++)
        {
            for (int k=0; k < rhs->num_rows; k++)
            {
                result->matrixAddr[i][j] += lhs->matrixAddr[i][k] * rhs->matrixAddr[k][j];
            }
        }
    }
    return result;
}

matrix* transpose(matrix* input) {
    int rows = input->num_cols;
    int cols = input->num_rows;

    int** matrixValues = malloc(cols * sizeof(int*));

    for (int i = 0; i < rows; i++) {
        int* matrix_col = malloc(rows * sizeof(int));
        *(matrixValues + i) = matrix_col;
        for (int j = 0; j < cols; j++) {
            matrix_col[j] = *(*((input->matrixAddr) + j)+i);
        }
    }

    input->num_rows = rows;
    input->num_cols = cols;
    input->matrixAddr = matrixValues;

    return input;
}


matrix* matrixMult(matrix* lhs, matrix* rhs) {
  //check dimensions
  if (lhs->num_rows != rhs->num_rows || lhs->num_cols != rhs->num_rows) {
    die("matrix add size mismatch");
  }
  int rows = lhs->num_rows;
  int cols= lhs->num_cols;
  matrix *result = initMatrix(NULL, rows, cols);
  for(int i=0; i<rows; i++) {
    for(int j=0; j<cols; j++) {
        int product = lhs->matrixAddr[i][j] * rhs->matrixAddr[i][j];
        result->matrixAddr[i][j] = product;
    }
  }

  return result;
}


matrix* timesScalar(matrix* input, int scalar) {
```

```c
  int rows = input->num_rows;
  int cols= input->num_cols;
  matrix *result = initMatrix(NULL, rows, cols);
  for(int i=0; i<rows; i++) {
    for(int j=0; j<cols; j++) {
        int product = input->matrixAddr[i][j] * scalar;
        result->matrixAddr[i][j] = product;
    }
  }

  return result;
}


void display(matrix* input) {
    int row = input->num_rows;
    int col = input->num_cols;
    for(int i = 0; i<row; i++) {
        for(int j=0; j<col; j++) {
            printf(" %d", input->matrixAddr[i][j]);
        }
        printf("\n");
    }
}

int string_length(char *s) {
    return strlen(s);
}

char *string_get(char *s, int i) {
    char *c = malloc(2);
    c[0] = s[i];
    c[1] = '\0';
    return c;
}

char *string_concat(char *s1, char *s2) {
    char *new = (char *) malloc(strlen(s1) + strlen(s2) + 1);
    strcpy(new, s1);
    strcat(new, s2);
    return new;
}

int string_equals(char *s1, char *s2) {
    return (strcmp(s1, s2) == 0);
}

char *string_substr(char *s, int start, int end) {
  char *substr = malloc(end - start+1);
    int i;
    for(i = 0; i < (end - start); i++) {
        substr[i] = s[start + i];
    }
    substr[end-start]=0;
    return substr;
}

#ifdef BUILD_TEST
int main(int argc,char** argv) {
  //run tests of each function
  //initMatrix and display of empty matrix
  matrix *null_matrix=initMatrix(NULL, 2, 2);
```

```
    printf("NULL MATRIX: \n");
    display(null_matrix);




    //initMatrix and display of 2x2 matrix
    int vals1[] = {3, 8, 4, 6};
    int *list1 = vals1;
    matrix *m = initMatrix(list1, 2, 2);
    printf("2x2 MATRIX: \n");
    display(m);

    //TODO test codegen builder
    for( int i = 0; i < 4; i++) {
        m = storeVal(m, 5);
         printf("Stroring 5: \n");
         display(m);
    }


    //add 2 of the same matrix
    matrix *result_sum = mAdd(m, m);
    printf("ADD TWO OF THE ORIGINAL 2x2 MATRIX: \n");
    display(result_sum);

    //multiply two matrices
    matrix *result_product = matrixMult(m, m);
    printf("MULTIPLY TWO OF THE ORIGINAL 2X2 MATRIX: \n");
    display(result_product);

    //scalar multiplication
    matrix *result_scalar = timesScalar(m, 3);
    printf("SCALAR MULTIPLICATION OF ORIGINAL MATRIX BY 3: \n");
    display(result_scalar);

    //determinant of 2x2 matrix
    printf("The determinant is %d\n", determinant(m, 2));

    //determinant of 3x3 matrix
    int vals2[] = {6, 1, 1, 4, -2, 5, 2, 8, 7};
    int *list2 = vals2;
    matrix *n = initMatrix(list2, 3, 3);
    printf("3x3 MATRIX: \n");
    display(n);
    printf("The determinant is %d\n", determinant(n, 3));

    //dot product tests
    int values_1[4] = {1,2,3,4};
    matrix* m1 = initMatrix(&values_1[0], 2, 2);
    int values_2[4] = {5,6,7,8};
    matrix* m2 = initMatrix(&values_2[0], 2, 2);

    printf("The dot product of matrices 1 and 2 is %d\n", dotProduct(m1,m2));

    transpose(m1);
    display(m1);
}
#endif
```

8.1.10 Makefile

```
# "make test" Compiles everything and runs the regression tests

.PHONY : test
test : all testall.sh
	./testall.sh

# "make all" builds the executable
# to test linking external code

.PHONY : all
all : matrx.native matrix.o

# "make matrx.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by including
# packages, enabling warnings
#
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

matrx.native : matrix.bc
	opam config exec -- \
	ocamlbuild -use-ocamlfind matrx.native -pkgs llvm,llvm.analysis,llvm.bitreader

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log ocamlllvm *.diff *.ll *.o *.bc matrix

#build the matrix file

matrix : matrix.c
	cc -o matrix -DBUILD_TEST matrix.c

matrix.bc : matrix.c
	clang -emit-llvm -o matrix.bc -c matrix.c -Wno-varargs

# Building the tarball

TESTS = \
  add1 arith1 arith2 arith3 fib float1 float2 float3 for1 for2 func1 \
  func2 func3 func4 func5 func6 func7 func8 func9 gcd2 gcd global1 \
  global2 global3 hello if1 if2 if3 if4 if5 if6 local1 local2 ops1 \
  ops2 var1 var2 while1 while2

FAILS = \
  assign1 assign2 assign3 dead1 dead2 expr1 expr2 expr3 float1 float2 \
  for1 for2 for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 \
  func8 func9 global1 global2 if1 if2 if3 nomain printb print \
  return1 return2 while1 while2
```

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

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags matrx.ml parser.mly \
	README scanner.mll semant.ml testall.sh \
	arcade-font.pbm font2c \
	Dockerfile \
	$(TESTFILES:%=tests/%)

matrx.tar.gz : $(TARFILES)
	cd .. && tar czf matrx/matrx.tar.gz \
```

8.1.11 README

```
The Matrx compiler

Coded in OCaml, this takes a highly stripped-down subset of C (ints,
bools, and void types, arithmetic, if-else, for, and while statements,
and user-defined functions) and compiles it into LLVM IR.

It needs the OCaml llvm library, which is most easily installed through opam.

Install LLVM and its development libraries, the m4 macro preprocessor,
and opam, then use opam to install llvm.

The version of the OCaml llvm library must match the version of the LLVM
system installed on your system.

In addition to print, which calls the C library function printf(),
Matrx gratuitiously includes a primitive function "printm," which
prints large ASCII-encoded characters.

The stock C compiler compiles matrix.o. testall.sh runs the Matrx
executable on each testcase (.mx file) to produce a .ll file, invokes
"llc" (the LLVM compiler) to produce a .s (assembly) file, then
invokes "cc" (the stock C compiler) to assemble the .s file, link in
matrix.o, and generate an executable. See testall.sh for details.
------------
If you get errors about llvm.analysis not being found, it's probably
because opam enviroment information is missing. Either run

eval $(opam config env)

or run ocamlbuild like this:

opam config exec -- ocamlbuild <args>
------------------------------
Using Docker

* Install Docker on whatever operating system you're on

  Under Ubuntu,
```

```
    apt install docker.io

* Test your installation

    docker run hello-world

   If this fails, you will need to correct your installation.

   Under Ubuntu, add yourself to the "docker" group:

   sudo usermod -aG docker <username>

* Move to where you unpacked the Matrx source:

   cd Matrx

* Invoke docker

   docker run --rm -it -v `pwd`:/home/Matrx -w=/home/Matrx columbiasedwards/plt

* Inside docker, compile Matrx and run the regression tests:

   # make
   ...
   test-add1...OK
   test-arith1...OK
   test-arith2...OK
   test-arith3...OK
   ... etc.

   # make clean

------------------------------
 Installation under Ubuntu 16.04

LLVM 3.8 is the default under 16.04. Install the matching version of
the OCaml LLVM bindings:

sudo apt install ocaml llvm llvm-runtime m4 opam
opam init
opam install llvm.3.8
eval `opam config env`

make
./testall.sh

------------------------------
 Installation under Ubuntu 15.10

LLVM 3.6 is the default under 15.10, so we ask for a matching version of the
OCaml library.

sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.6 ocamlfind
eval `opam config env`

make
./testall.sh
```

```

---------------------------------------
Installation under Ubuntu 14.04

The default LLVM package is 3.4, so we install the matching OCaml
library using opam.  The default version of opam under 14.04 is too
old; we need to use a newer package.

sudo apt-get install m4 llvm software-properties-common

sudo add-apt-repository --yes ppa:avsm/ppa
sudo apt-get update -qq
sudo apt-get install -y opam
opam init

eval `opam config env`

opam install llvm.3.4 ocamlfind

---------------------------------------
Installation under OS X

1. Install Homebrew:

   ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

2. Verify Homebrew is installed correctly:

   brew doctor

3. Install opam:

   brew install opam

4. Set up opam:

   opam init

5. Install llvm:

   brew install llvm

   Take note of where brew places the llvm executables. It will show
   you the path to them under the CAVEATS section of the post-install
   terminal output. For me, they were in /usr/local/opt/llvm/bin. Also
   take note of the llvm version installed. For me, it was 3.6.2.

6. Have opam set up your enviroment:

   eval `opam config env`

7. Install the OCaml llvm library:

   opam install llvm.3.6

   Ensure that the version of llvm you install here matches the
   version you installed via brew. Brew installed llvm version 3.6.2,
```

```
   so I install llvm.3.6 with opam.

   IF YOU HAVE PROBLEMS ON THIS STEP, it's probably because you are
   missing some external dependencies. Ensure that libffi is installed
   on your machine. It can be installed with

   brew install libffi

   If, after this, opam install llvm.3.6 is still not working, try
   running

   opam list --external --required-by=llvm.3.6

   This will list all of the external dependencies required by
   llvm.3.6. Install all the dependencies listed by this command.

   IF THE PREVIOUS STEPS DO NOT SOLVE THE ISSUE, it may be a problem
   with using your system's default version of llvm. Install a
   different version of llvm and opam install llvm with that version
   by running:

   brew install homebrew/versions/llvm37
   opam install llvm.3.7

   Where the number at the end of both commands is a version different
   from the one your system currently has.

8. Make sure testall.sh can access lli and llc

   Modify the definition of LLI and LLC in testall.sh to point to the absolute
   path, e.g., LLI="/usr/local/opt/llvm/bin/lli"

   - OR -

   Update your path, e.g.,

   export PATH=$PATH:/usr/local/opt/llvm/bin

   - OR -

   Create a symbolic link to the lli command:

   sudo ln -s /usr/local/opt/llvm/bin/lli /usr/bin/lli

   Create the symlink from wherever brew installs the llvm executables
   and place it in your bin. From step 5, I know that brew installed
   the lli executable in the folder, /usr/local/opt/llvm/bin/, so this
   is where I symlink to. Brew might install the lli executables in a
   different location for you, so make sure you symlink to the right
   directory.
```

```
   IF YOU GET OPERATION NOT PERMITTED ERROR, then this is probably a
   result of OSX's System Integrity Protection.

   One way to get around this is to reboot your machine into recovery
   mode (by holding cmd-r when restarting). Open a terminal from
   recovery mode by going to Utilities -> Terminal, and enter the
   following commands:

   csrutil disable
   reboot

   After your machine has restarted, try the `ln....` command again,
   and it should succeed.

   IMPORTANT: the prevous step disables System Integrity Protection,
   which can leave your machine vulnerable. It's highly advisable to
   reenable System Integrity Protection when you are done by
   rebooting your machine into recovery mode and entering the following
   command in the terminal:

   csrutil enable
   reboot

9. To run and test, navigate to the Matrx folder. Once there, run

   make ; ./testall.sh

   Matrx should build without any complaints and all tests should
   pass.

   IF RUNNING ./testall.sh FAILS ON SOME TESTS, check to make sure you
   have symlinked the correct executable from your llvm installation.
   For example, if the executable is named lli-[version], then the
   previous step should have looked something like:

   sudo ln -s /usr/local/opt/llvm/bin/lli-3.7 /usr/bin/lli

   As before, you may also modify the path to lli in testall.sh

------------------------------
To run and test:

$ make
ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 Matrx.native
Finished, 22 targets (0 cached) in 00:00:01.
cc    -c -o matrix.o matrix.c
$ ./testall.sh
test-arith1...OK
test-arith2...OK
test-arith3...OK
test-fib...OK
...
fail-while1...OK
```

```
fail-while2...OK
```

8.1.12 demo.mx

```
int main() {
  matrix a = [[1,2]];
  matrix b;
  matrix c = [[1,2],[3,4]];
  matrix d = [[4,4],[4,4]];
  matrix e;
  matrix f = [[1][2][3][4][5][6]];

  printstr("  matrix a = [[1,2]];
  matrix b;
  matrix c = [[1,2],[3,4]];
  matrix d = [[4,4],[4,4]];
  matrix e;
  matrix f = [[1][2][3][4][5][6]];
  ");

  b = [[3,4]];
  printstr("Print Matrix A:");
  printm(a);
  printstr("");
  printstr("Print Matrix B:");
  printm(b);

  printstr("");

  printstr("Transpose Matrix B:");
  printm(transpose(b));

  printstr("");

  printstr("Dot A and B");
  printm(dot(a,b));

  printstr("");
  printstr("Dot B and A");
  printm(dot(b,a));

  printstr("");
  printstr("[[1,2,3][3,3,3]] DOT [[2][3][1]]");
  printm(dot([[1,2,3][3,3,3]],[[2][3][1]]));

  printstr("");
  printstr("Print Matrix C:");
  printm(c);
  printstr("");

  printstr("Print Matrix D:");
  printm(d);
  printstr("");

  printstr("E = C + D:");
  printm(e = matadd(c, d));
  printstr("");

  printstr("C * D:");
  printm(matmult(c, d));
  printstr("");
```

```
  printstr("Det of [[1,0][0,1]]:");
  print(det([[1,0][0,1]],2));
  printstr("");

  printstr("Det of [[1,2][3,4]]:");
  print(det([[1,2][3,4]],2));
  printstr("");

  printstr("Scale [[1,1][2,2]] by 3");
  printm(matscale([[1,1][2,2]], 3));
  printstr("");

  return 0;
}
```

8.2 Tests

8.2.1 fail-assign1.err

```
Fatal error: exception Failure("illegal assignment int = bool in i = false")
```

8.2.2 fail-assign1.mx

```
int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
}
```

8.2.3 fail-assign2.err

```
Fatal error: exception Failure("illegal assignment bool = int in b = 48")
```

8.2.4 fail-assign2.mx

```
int main()
{
  int i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}
```

8.2.5 fail-assign3.err

```
Fatal error: exception Failure("illegal assignment int = void in i
    ↪ = myvoid()")
```

8.2.6 fail-assign3.mx

```
void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
```

8.2.7 fail-dead1.err

```
Fatal error: exception Failure("nothing may follow a return")
```

8.2.8 fail-dead1.mx

```
int main()
{
  int i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}
```

8.2.9 fail-dead2.err

```
Fatal error: exception Failure("nothing may follow a return")
```

8.2.10 fail-dead2.mx

```
int main()
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
}
```

8.2.11 fail-expr1.err

```
Fatal error: exception Failure("illegal binary operator bool + int
    ↪ in d + a")
```

8.2.12 fail-expr1.mx

```
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

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

8.2.13 fail-expr2.err

```
Fatal error: exception Failure("illegal binary operator bool + int
    ↪ in b + a")
```

8.2.14 fail-expr2.mx

```
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}

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

8.2.15 fail-expr3.err

```
Fatal error: exception Failure("illegal binary operator float + int
    ↪  in b + a")
```

8.2.16 fail-expr3.mx

```
int a;
float b;

void foo(int c, float d)
{
  int d;
  float e;
  b + a; /* Error: float + int */
}

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

8.2.17 fail-float1.err

```
Fatal error: exception Failure("illegal binary operator float &&
    ↪ int in -3.5 && 1")
```

8.2.18 fail-float1.mx

```
int main()
{
  -3.5 && 1; /* Float with AND? */
  return 0;
}
```

8.2.19 fail-float2.err

```
Fatal error: exception Failure("illegal binary operator float &&
    ↪ float in -3.5 && 2.5")
```

8.2.20 fail-float2.mx

```
int main()
{
  -3.5 && 2.5; /* Float with AND? */
  return 0;
}
```

8.2.21 fail-for1.err

```
Fatal error: exception Failure("undeclared identifier j")
```

8.2.22 fail-for1.mx

```
int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
```

8.2.23 fail-for2.err

```
Fatal error: exception Failure("undeclared identifier j")
```

8.2.24 fail-for2.mx

```
int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
```

8.2.25 fail-for3.err

```
Fatal error: exception Failure("expected Boolean expression in i")
```

8.2.26 fail-for3.mx

```
int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */

  return 0;
}
```

8.2.27 fail-for4.err

```
Fatal error: exception Failure("undeclared identifier j")
```

8.2.28 fail-for4.mx

```
int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}
```

8.2.29 fail-for5.err

```
Fatal error: exception Failure("unrecognized function foo")
```

8.2.30 fail-for5.mx

```
int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}
```

8.2.31 fail-func1.err

```
Fatal error: exception Failure("duplicate function bar")
```

8.2.32 fail-func1.mx

```
int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

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

8.2.33 fail-func2.err

```
Fatal error: exception Failure("duplicate formal a")
```

8.2.34 fail-func2.mx

```
int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in
    ↪ bar */

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

8.2.35 fail-func3.err

```
Fatal error: exception Failure("illegal void formal b")
```

8.2.36 fail-func3.mx

```
int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b
    ↪ */

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

8.2.37 fail-func4.err

```
Fatal error: exception Failure("function print may not be defined")
```

8.2.38 fail-func4.mx

```
int foo() {}

void bar() {}

int print() {} /* Should not be able to define print */

void baz() {}

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

8.2.39 fail-func5.err

```
Fatal error: exception Failure("illegal void local b")
```

8.2.40 fail-func5.mx

```
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

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

8.2.41 fail-func6.err

```
Fatal error: exception Failure("expecting 2 arguments in foo(42)")
```

8.2.42 fail-func6.mx

```
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}
```

8.2.43 fail-func7.err

```
Fatal error: exception Failure("expecting 2 arguments in foo(42,
    ↪ true, false)")
```

8.2.44 fail-func7.mx

```
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}
```

8.2.45 fail-func8.err

```
Fatal error: exception Failure("illegal argument found void
    ↪ expected bool in bar()")
```

8.2.46 fail-func8.mx

```
void foo(int a, bool b)
{
}

void bar()
{
}

int main()
{
  foo(42, true);
  foo(42, bar()); /* int and void, not int and bool */
}
```

8.2.47 fail-func9.err

```
Fatal error: exception Failure("illegal argument found int expected
    ↪ bool in 42")
```

8.2.48 fail-func9.mx

```
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}
```

8.2.49 fail-global1.err

```
Fatal error: exception Failure("illegal void global a")
```

8.2.50 fail-global1.mx

```
int c;
bool b;
void a; /* global variables should not be void */


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

8.2.51 fail-global2.err

```
Fatal error: exception Failure("duplicate global b")
```

8.2.52 fail-global2.mx

```
int b;
bool c;
int a;
int b; /* Duplicate global variable */

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

8.2.53 fail-if1.err

```
Fatal error: exception Failure("expected Boolean expression in 42")
```

8.2.54 fail-if1.mx

```
int main()
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}
```

8.2.55 fail-if2.err

```
Fatal error: exception Failure("undeclared identifier foo")
```

8.2.56 fail-if2.mx

```
int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}
```

8.2.57 fail-if3.err

```
Fatal error: exception Failure("undeclared identifier bar")
```

8.2.58 fail-if3.mx

```
int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}
```

8.2.59 fail-matrix-dims-3D.err

```
Fatal error: exception Parsing.Parse_error
```

8.2.60 fail-matrix-dims-3D.mx

```
int main(){
    matrix m;
    m = [
            [ [1, 2, 3], [3,4,5] ],
            [ [1, 2, 3], [3,4,5] ]
        ];
}
```

8.2.61 fail-matrix-dims.err

```
Fatal error: exception Failure("Invalid dims")
```

8.2.62 fail-matrix-dims.mx

```
int main(){
    matrix a;
    a = [ [1, 2, 3], [3,4,5,4] ];
}
```

8.2.63 fail-nomain.err

```
Fatal error: exception Failure("unrecognized function main")
```

8.2.64 fail-printb.err

```
Fatal error: exception Failure("function printb may not be defined ")
```

8.2.65 fail-printb.mx

```
/* Should be illegal to redefine */
void printb() {}
```

8.2.66 fail-print.err

```
Fatal error: exception Failure("function print may not be defined")
```

8.2.67 fail-print.mx

```
/* Should be illegal to redefine */
void print() {}
```

8.2.68 fail-return1.err

```
Fatal error: exception Failure("return gives bool expected int in true")
```

8.2.69 fail-return1.mx

```
int main()
{
  return true; /* Should return int */
}
```

8.2.70 fail-return2.err

```
Fatal error: exception Failure("return gives int expected void in 42")
```

8.2.71 fail-return2.mx

```
void foo()
{
  if (true) return 42; /* Should return void */
  else return;
}

int main()
{
  return 42;
}
```

8.2.72 fail-while1.err

```
Fatal error: exception Failure("expected Boolean expression in 42")
```

8.2.73 fail-while1.mx

```
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
    i = i + 1;
  }

}
```

8.2.74 fail-while2.err

```
Fatal error: exception Failure("unrecognized function foo")
```

8.2.75 fail-while2.mx

```
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}
```

8.2.76 test-add1.mx

```
int add(int x, int y)
{
  return x + y;
}

int main()
{
  print( add(17, 25) );
  return 0;
}
```

8.2.77 test-add1.out

```
42
```

8.2.78 test-arith1.mx

```
int main()
{
  print(39 + 3);
  return 0;
}
```

8.2.79 test-arith1.out

```
42
```

8.2.80 test-arith2.mx

```
int main()
{
  print(1 + 2 * 3 + 4);
  return 0;
}
```

8.2.81 test-arith2.out

```
11
```

8.2.82 test-arith3.mx

```
int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
}
```

8.2.83 test-arith3.out

```
47
```

8.2.84 test-fib.mx

```
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

int main()
{
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
  return 0;
}
```

8.2.85 test-fib.out

```
1
1
2
3
5
8
```

8.2.86 test-float1.mx

```
int main()
{
  float a;
  a = 3.14159267;
  printf(a);
  return 0;
}
```

8.2.87 test-float1.out

```
3.14159
```

8.2.88 test-float2.mx

```
int main()
{
  float a;
  float b;
  float c;
  a = 3.14159267;
  b = -2.71828;
  c = a + b;
  printf(c);
  return 0;
}
```

8.2.89 test-float2.out

```
0.423313
```

8.2.90 test-float3.mx

```
void testfloat(float a, float b)
{
  printf(a + b);
  printf(a - b);
  printf(a * b);
  printf(a / b);
  printb(a == b);
  printb(a == a);
  printb(a != b);
  printb(a != a);
  printb(a > b);
  printb(a >= b);
  printb(a < b);
  printb(a <= b);
}

int main()
{
  float c;
  float d;

  c = 42.0;
  d = 3.14159;

  testfloat(c, d);

  testfloat(d, d);

  return 0;
}
```

8.2.91 test-float3.out

```
45.1416
38.8584
131.947
13.369
0
1
1
0
1
1
0
0
6.28318
0
9.86959
1
1
1
0
```

```
0
0
1
0
1
```

8.2.92 test-for1.mx

```
int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}
```

8.2.93 test-for1.out

```
0
1
2
3
4
42
```

8.2.94 test-for2.mx

```
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}
```

8.2.95 test-for2.out

```
0
1
2
3
4
42
```

8.2.96 test-func1.mx

```
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}
```

8.2.97 test-func1.out

```
42
```

8.2.98 test-func2.mx

```
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);
  return 0;
}
```

8.2.99 test-func2.out

```
2
```

8.2.100 test-func3.mx

```
void printem(int a, int b, int c, int d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}

int main()
```

```
{
  printem(42,17,192,8);
  return 0;
}
```

8.2.101 test-func3.out

```
42
17
192
8
```

8.2.102 test-func4.mx

```
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}
```

8.2.103 test-func4.out

```
62
```

8.2.104 test-func5.mx

```
int foo(int a)
{
  return a;
}

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

8.2.105 test-func6.mx

```
void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
```

```
{
  print(bar(17, false, 25));
  return 0;
}
```

8.2.106 test-func6.out

```
42
```

8.2.107 test-func7.mx

```
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}
```

8.2.108 test-func7.out

```
115
```

8.2.109 test-func8.mx

```
void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
}
```

8.2.110 test-func8.out

```
43
```

8.2.111 test-func9.mx

```
void foo(int a)
{
  print(a + 3);
  return;
}

int main()
{
  foo(40);
  return 0;
}
```

8.2.112 test-func9.out

```
43
```

8.2.113 test-gcd2.mx

```
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

int main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}
```

8.2.114 test-gcd2.out

```
7
4
11
```

8.2.115 test-gcd.mx

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  print(gcd(2,14));
```

```
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}
```

8.2.116 test-gcd.out

```
2
3
11
```

8.2.117 test-global1.mx

```
int a;
int b;

void printa()
{
  print(a);
}

void printbb()
{
  print(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printbb();
  incab();
  printa();
  printbb();
  return 0;
}
```

8.2.118 test-global1.out

```
42
21
43
22
```

8.2.119 test-global2.mx

```
bool i;

int main()
{
  int i; /* Should hide the global i */

  i = 42;
  print(i + i);
  return 0;
}
```

8.2.120 test-global2.out

```
84
```

8.2.121 test-global3.mx

```
int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}
```

8.2.122 test-global3.out

```
52
```

8.2.123 test-hello.mx

```
int main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}
```

8.2.124 test-hello.out

```
42
71
1
```

8.2.125 test-if1.mx

```
int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
```

8.2.126 test-if1.out

```
42
17
```

8.2.127 test-if2.mx

```
int main()
{
  if (true) print(42); else print(8);
  print(17);
  return 0;
}
```

8.2.128 test-if2.out

```
42
17
```

8.2.129 test-if3.mx

```
int main()
{
  if (false) print(42);
  print(17);
  return 0;
}
```

8.2.130 test-if3.out

```
17
```

8.2.131 test-if4.mx

```
int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}
```

8.2.132 test-if4.out

```
8
17
```

8.2.133 test-if5.mx

```
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
```

8.2.134 test-if5.out

```
42
17
```

8.2.135 test-if6.mx

```
int cond(bool b)
{
  int x;
  x = 10;
  if (b)
    if (x == 10)
      x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
```

8.2.136 test-if6.out

```
42
10
```

8.2.137 test-local1.mx

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

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

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

8.2.138 test-local1.out

```
84
```

8.2.139 test-local2.mx

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

  c = a;

  return c + 10;
}

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

8.2.140 test-local2.out

```
47
```

8.2.141 test-matadd.mx

```
int main() {
    matrix m;
    m = [[1, 2], [3,4] ];
    printm(matadd(m,m));
}
```

8.2.142 test-matadd.out

```
 2 4
 6 8
```

8.2.143 test-matdet.mx

```
int main() {
    matrix m;
    m = [[1, 2], [3,4] ];
    print(det(m,2));
}
```

8.2.144 test-matdet.out

```
-2
```

8.2.145 test-mat-dot2.mx

```
int main() {
    matrix m;
    matrix n = [[1][3]];
    m = [[1, 2]];
    printm(dot(n,m));
}
```

8.2.146 test-mat-dot2.out

```
 1 2
 3 6
```

8.2.147 test-mat-dot.mx

```
int main() {
    matrix m;
    matrix n = [[1][3]];
    m = [[1, 2]];
    printm(dot(m,n));
}
```

8.2.148 test-mat-dot.out

```
 7
```

8.2.149 test-matmult.mx

```
int main() {
    matrix m;
    m = [[1, 2], [3,4] ];
    printm(matmult(m,m));
}
```

8.2.150 test-matmult.out

```
 1 4
 9 16
```

8.2.151 test-mat-print2.mx

```
int main() {
    matrix m;
    m = [[1, 2], [3,4],[5,6]];
    printm(m);
}
```

8.2.152 test-mat-print2.out

```
 1 2
 3 4
 5 6
```

8.2.153 test-mat-print.mx

```
int main() {
    matrix m;
    m = [[1, 2][3,4] ];
    printm(m);
}
```

8.2.154 test-mat-print.out

```
 1 2
 3 4
```

8.2.155 test-matrix1.mx

```
int main(){
    matrix a = [ [1, 2] [3, 4] ];
    printm(a);
    return 0;
}
```

8.2.156 test-matrix1.out

```
 1 2
 3 4
```

8.2.157 test-matscale.mx

```
int main() {
     matrix m;
    m = [[1,  2],  [3,4]  ];
     printm(matscale(m,4));
}
```

8.2.158 test-matscale.out

```
 4 8
 12 16
```

8.2.159 test-mat-trans.mx

```
int main() {
     matrix m;
    m = [[1,  2],  [3,4]  ];
     printm(transpose(m));
}
```

8.2.160 test-mat-trans.out

```
 1 3
 2 4
```

8.2.161 test-ops1.mx

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

```
    return 0;
}
```

8.2.162 test-ops1.out

```
3
-1
2
50
99
0
1
99
1
0
99
1
0
99
1
1
0
99
0
1
99
0
1
1
```

8.2.163 test-ops2.mx

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

8.2.164 test-ops2.out

```
1
0
1
0
```

```
0
0
1
1
1
0
1
0
−10
42
```

8.2.165 test-string1.mx

```
int main()
{
  string s1 = "This is an example";
  printstr(get_char(s1, 0));
  return 0;
}
```

8.2.166 test-string1.out

```
T
```

8.2.167 test-string2.mx

```
int main()
{
  printb(sequals("MATRX", "MATRX"));
  printstr(sconcat("This is an example ", "of a string in MATRX!"))
      ↪ ;
  printstr(substring("This is an example of a string in MATRX!", 3,
      ↪  7));
  return 0;
}
```

8.2.168 test-string2.out

```
1
This is an example of a string in MATRX!
s is
```

8.2.169 test-string-decl1.mx

```
int main()
{
  string s1 = "This is an example";
  printstr(s1);

  return 0;
}
```

8.2.170 test-string-decl1.out

```
This is an example
```

8.2.171 test-string-decl2.mx

```
int main()
{
  string s2;
  s2 = "This is another example";
  printstr(s2);

  return 0;
}
```

8.2.172 test-string-decl2.out

```
This is another example
```

8.2.173 test-var1.mx

```
int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}
```

8.2.174 test-var1.out

```
42
```

8.2.175 test-var2.mx

```
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}
```

8.2.176 test-var2.out

```
115
```

8.2.177 test-while1.mx

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```

8.2.178 test-while1.out

```
5
4
3
2
1
42
```

8.2.179 test-while2.mx

```
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  print(foo(7));
  return 0;
}
```

8.2.180 test-while2.out

```
14
```